

**An aspect-oriented approach towards enhancing Optimistic Access Control
with Usage Control**

by

Keshnee Padayachee

submitted in fulfilment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in the subject of

COMPUTER SCIENCE

in the

Faculty of Engineering, Built Environment and Information Technology

at the

UNIVERSITY OF PRETORIA

SUPERVISOR: Prof. J.H.P. Eloff

DECEMBER 2009



PREFACE

This research was conducted on a part-time basis between 2004 and 2009 in collaboration with the Department of Computer Science at the University of Pretoria under the supervision of Professor J.H.P. Eloff. The results are the original work of the author and have not been submitted for any degree at any other tertiary institution.



ABSTRACT

With the advent of agile programming, lightweight software processes are being favoured over the highly formalised approaches of the 80s and 90s, where the emphasis is on "people, not processes". Likewise, access control may benefit from a less prescriptive approach and an increasing reliance on users to behave ethically. These ideals correlate with optimistic access controls. However, such controls alone may not be adequate as they are retrospective rather proactive. Optimistic access controls may benefit from the stricter enforcement offered by usage control. The latter enables finer-grained control over the usage of digital objects than do traditional access control policies and models, as trust management concerns are also taken into consideration. This thesis investigates the possibility of enhancing optimistic access controls with usage control to ensure that users conduct themselves in a trustworthy manner. Since this kind of approach towards access control has limited applicability, the present study investigates contextualising this approach within a mixed-initiative access control framework. A mixed-initiative access control framework involves combining a minimum of two access control models where the request to information is mediated by a mixture of access policy enforcement agents. In order for this type of integration to be successful, a software development approach was considered that allows for the seamless augmentation of traditional access control with optimistic access control enhanced with usage control, namely the aspect-oriented approach. The aspect-oriented paradigm can facilitate the implementation of additional security features to legacy systems without modifying existing code. This study therefore evaluates the aspect-oriented approach in terms of implementing security concerns.

It is evidently difficult to implement access control and in dynamic environments preconfigured access control policies may often change dramatically, depending on the context. In unpredicted circumstances, users who are denied access could often have prevented a catastrophe had they been allowed access. The costs of implementing and maintaining complex preconfigured access control policies sometimes far outweigh the benefits. Optimistic controls are retrospective and allow users to exceed their normal privileges. However, if a user accesses information unethically, the consequences could be



disastrous. Therefore it is proposed that optimistic access control be enhanced with some form of usage control, which may prevent the user from engaging in risky behaviour.

An initiative towards including security in the earlier phases of the software life cycle is gaining momentum, as it is much easier to design with security from the onset than to use the penetrate-and-patch approach. Unfortunately, incorporating security into software development takes time and developers tend to focus more on the features of the software application. The aspect-oriented paradigm can facilitate the implementation of additional security features in legacy systems without modifying existing code. The current study evaluates the aspect-oriented approach towards enhancing optimistic access control with usage control. The efficacy of the aspect-oriented paradigm has been well established within several areas of software security, as aspect-orientation facilitates the abstraction of these security-related tasks so as to reduce code complexity.



SUMMARY

Title: An aspect-oriented approach towards enhancing Optimistic Access Control with Usage Control

Candidate: Keshnee Padayachee

Supervisor: J.H.P. Eloff

Department: Department of Computer Science, Faculty of Engineering, Built Environment and Information Technology

Degree: Doctor of Philosophy in Computer Science

Keywords: Usage Control, Optimistic Access Control, Access Control, Aspect-Oriented Programming



ACKNOWLEDGEMENTS

This work would not have been possible without the support and encouragement of my supervisor Professor J.H.P. Eloff.

I am grateful to my husband Devern Padayachee; my nephews Ryan and André Veerasamy; and my colleagues at UNISA, especially Professor Elmé Smith for giving me the requisite courage to prevail. I also give credit to providence for giving me the perseverance to continue in spite of the odds.

It always seems impossible until is done
-Nelson Mandela



TABLE OF CONTENTS

| | |
|--|-----------|
| PART 1: | 1 |
| CHAPTER 1: INTRODUCTION | 2 |
| 1.1 Introduction | 2 |
| 1.2 Motivation for this study | 5 |
| 1.3 Problem Statement | 9 |
| 1.4 Terminology used in this thesis..... | 9 |
| 1.5 Research Methodology | 10 |
| 1.6 Delimitations | 11 |
| 1.7 Thesis Layout | 11 |
| 1.8 Summary | 14 |
| CHAPTER 2: ACCESS CONTROL | 15 |
| 2.1 Introduction | 15 |
| 2.2 Discretionary Access Control | 15 |
| 2.3 Mandatory Access Control..... | 17 |
| 2.4 Role-based Access Control..... | 19 |
| 2.5 Conclusion | 20 |
| CHAPTER 3: OPTIMISTIC ACCESS CONTROL | 22 |
| 3.1 Introduction | 22 |
| 3.2 Optimistic Access Control | 23 |
| 3.3 Requirements for Optimistic Security | 24 |
| 3.4 Applicability of optimistic security..... | 25 |
| 3.5 The extensibility of the Optimistic Access Control Model | 26 |
| 3.6 Conclusion | 29 |



| | | |
|-------------------|---|-----------|
| CHAPTER 4: | USAGE CONTROL..... | 30 |
| 4.1 | Introduction | 30 |
| 4.2 | The continuity and mutability of the UCON model | 31 |
| 4.3 | The ABC Model for Usage Control (UCON model) | 32 |
| 4.4 | The Usage Control Model architecture | 34 |
| 4.5 | The Applicability and Extensibility of the UCON model..... | 36 |
| 4.6 | Conclusion | 37 |
| PART 2: | | 39 |
| CHAPTER 5: | ASPECT-ORIENTED PROGRAMMING..... | 40 |
| 5.1 | Introduction | 40 |
| 5.2 | Evolution to Aspect-Oriented Programming | 41 |
| 5.3 | Aspect-Oriented Programming Terminology..... | 43 |
| 5.4 | AOP Frameworks | 44 |
| 5.5 | Evaluating Aspect-Oriented Programming..... | 46 |
| 5.6 | Conclusion | 48 |
| CHAPTER 6: | ASPECT-ORIENTED SECURITY..... | 49 |
| 6.1 | Introduction | 49 |
| 6.2 | Aspect-oriented programming and its application to security | 50 |
| 6.2.1 | Access Control and Authentication | 50 |
| 6.2.2 | Accountability and Audit | 52 |
| 6.2.3 | Cryptographic Controls..... | 52 |
| 6.2.4 | Information Flow Controls..... | 53 |
| 6.2.5 | Protection from invasive software | 53 |
| 6.2.6 | Security kernels..... | 54 |
| 6.2.7 | Verification | 54 |
| 6.3 | Conclusion | 55 |



| | |
|--|------------|
| PART 3: | 56 |
| CHAPTER 7: THE OAC(UCON) MODEL | 57 |
| 7.1 Introduction | 57 |
| 7.2 A motivating example..... | 57 |
| 7.3 Architecture..... | 59 |
| 7.4 Detailed Design | 64 |
| 7.4.1 Formal Specifications..... | 64 |
| 7.4.2 The Use Case Diagram of Usage Control under the Optimistic Access Control Paradigm | 67 |
| 7.5 Conclusion | 69 |
| CHAPTER 8: PROTOTYPING AND MODEL EVALUATION | 70 |
| 8.1 Introduction | 70 |
| 8.2 The aim of the proof-of-concept prototype | 70 |
| 8.3 Implementation of the proof-of-concept prototype | 71 |
| 8.4 An implementation overview of the proof-of-concept prototype..... | 76 |
| 8.5 Proof-of-concept prototype operation | 80 |
| 8.6 Evaluation of the Aspect-Oriented Approach..... | 83 |
| 8.6.1 The Design Approach | 83 |
| 8.6.2 Execution Time and Memory Usage | 85 |
| 8.7 Evaluation of the model concept | 87 |
| 8.8 Conclusion | 93 |
| CHAPTER 9: CONCLUSION | 95 |
| 9.1 Introduction | 95 |
| 9.2 Main contribution..... | 95 |
| 9.3 Revisiting the problem statement | 97 |
| 9.4 Future Research Directions | 98 |
| 9.5 Conclusion | 99 |
| REFERENCES | 100 |
| INDEX | 116 |



APPENDICES

| | |
|---|-----|
| Appendix A: List of Publications..... | 118 |
| Appendix B: OOP Documentation..... | 122 |
| Appendix C: AOP Documentation | 138 |
| Appendix D: Prototype Evaluation..... | 157 |
| Appendix E: Data Collection..... | 165 |
| Appendix F: AspectJ Semantics | 171 |
| Appendix G: Running the Demo Project..... | 172 |



LIST OF FIGURES

| | |
|--|----|
| <i>Figure 1-1: Overview of Thesis</i> | 13 |
| <i>Figure 2-1: Discretionary Access Control based on an Access Control List (adapted from(Tolone et al., 2005))</i> .. | 16 |
| <i>Figure 2-2: Mandatory access control (MAC) (adapted from (Russell and Gangemi, 1991))</i> | 17 |
| <i>Figure 2-3: Role-based Access Control (adapted from (Samarati and de Capitani di Vimercati, 2001))</i> | 19 |
| <i>Figure 4- 1: Continuity and Mutability Properties(Park et al.)</i> | 31 |
| <i>Figure 4-2: ABC Model Components(Sandhu and Park, 2003)</i> | 32 |
| <i>Figure 4-3: Conceptual Structure for the UCON Reference Monitor (Sandhu and Park, 2003)</i> | 35 |
| <i>Figure 5-1: Illustration of the Weaving Concept</i> | 44 |
| <i>Figure 7-1: Architectural Diagram</i> | 59 |
| <i>Figure 7-2: Conceptual Structure for Optimistic Access Control enhanced with Usage Control</i> | 61 |
| <i>Figure 7-3: A Mixed-Initiative Access Control Framework – combining RBAC with OAC(UCON)</i> | 64 |
| <i>Figure 7-4:Use Case Diagram of OAC(UCON)</i> | 67 |
| <i>Figure 8-1: State Activity diagram of OAC(UCON) Model)</i> | 71 |
| <i>Figure 8-2: Thread Diagram of the OAC(UCON) model</i> | 72 |
| <i>Figure 8-3: UML Diagram showing Aspect UsageControlInjector and Core Classes</i> | 75 |
| <i>Figure 8-4: Showing the OOP UML of Core Classes</i> | 84 |
| <i>Figure 8-5: OOP package level diagram vs AOP package level diagram (on the right)</i> | 85 |
| <i>Figure 8-6: Showing comparisons of the execution time of OO vs AOP</i> | 86 |
| <i>Figure 8-7: Showing comparisons of and Memory Usage of OO vs AOP</i> | 86 |



PROGRAM LISTINGS

| | |
|---|----|
| <i>Program Listing 6-1: Generalised Aspect Code for Access Control</i> | 51 |
| <i>Program Listing 6-2: Demonstrating Accountability and Auditing with Aspect-Orientation</i> | 52 |
| <i>Program Listing 8-1: 'SampleAuthorization' class</i> | 73 |
| <i>Program Listing 8-2: Showing the UsageControlInjector Aspect</i> | 77 |
| <i>Program Listing 8-3: Depicting an InterTypeDeclaration Aspect</i> | 79 |



PART 1:

**CHAPTER 1:
INTRODUCTION**

**CHAPTER 2:
ACCESS CONTROL MODELS**

**CHAPTER 3:
OPTIMISTIC ACCESS CONTROL**

**CHAPTER 4:
USAGE CONTROL**

CHAPTER 1:

INTRODUCTION

1.1 Introduction

With the advent of agile programming, lightweight software processes are being favoured over the highly formalised approaches of the 80s and 90s, where the emphasis is on people, not processes (Boehm, 2002). Likewise, access control may benefit from a less prescriptive approach with an increasing reliance on users to behave ethically. These ideals correlate with optimistic access controls. However, such controls alone may not be enough to ensure that users behave in a trustworthy manner. This research presents a model for enhancing optimistic access controls with usage control to ensure that users conduct themselves in a trustworthy manner. Usage control enables finer-grained control over the usage of digital objects than do traditional access control policies and models, as trust management concerns are also considered. It has become evident that the means by which software is designed and implemented can have a significant impact on software security (Devanbu and Stubblebine, 2000). The aspect-oriented paradigm can facilitate the implementation of additional security features to legacy systems without modifying existing code. This study therefore evaluates the aspect-oriented approach in terms of implementing security concerns such as usage control.

It is evidently difficult to implement access control and often in dynamic environments preconfigured access control policies may change dramatically depending on the context. Often in unpredicted circumstances users that are denied access could have prevented a catastrophe had they been allowed access. Consider as an example, a nurse – at a hospital that has been isolated during a tornado – who needs access to a patient's records but cannot access them as nurses are not authorised to access this information (Povey, 1999).

In this extreme case, it is possible that the patient's health and safety may be unnecessarily comprised due to the restrictions imposed by the access control system. The costs of implementing and maintaining complex preconfigured access control policies sometimes far outweigh the benefits. Optimistic access controls are retrospective and allow users to exceed their normal privileges. However, if a user accesses information unethically, the consequences could be disastrous. Hence this research proposes that optimistic access control be enhanced with some form of usage control that may prevent the user from engaging in risky behaviour.

Sandhu and Park (2003) who recognised the inadequacy of traditional access control models, proposed a new approach to access control called Usage Control (UCON). This model encompasses emerging applications such as trust management, in a unified framework. They claim that the missing components of traditional access control are the concepts of *obligations* and *conditions*. *Obligations* require some action by the subject so as to gain or sustain access, e.g. by clicking the ACCEPT button on a licence agreement. *Conditions* represent system-oriented factors such as time-of-day, where subjects are allowed access only within a specific time period. A family of models for usage control exists, involving pre-authorisation and ongoing authorisations.

The openness and flexibility of the optimistic access control approach has limited applicability. Hence this study investigates contextualising this approach within a mixed-initiative access control framework. According to Dewan et al. (2007), the mixed initiative access control approach is a means to resolve situations where users may wish to provide different controls for different objects or where users wish to have preferences in terms of their privacy settings. Such a control framework involves combining a minimum of two access control models where the request for information then is mediated by a mixture of access policy enforcement agents. In order for this type of integration to be successful, a software development approach was considered that would allow for the seamless augmentation of traditional access control with optimistic access control enhanced with usage control. Such an approach was found to be the aspect-oriented approach. The aspect-oriented paradigm can facilitate the implementation of additional security features to legacy systems without

modifying existing code. Consequently this study evaluates the aspect-oriented approach in terms of implementing security concerns.

Security is often extracted as a separable concern, due to its orthogonal nature in respect of the functional requirements of a system. Hence the separation-of-concerns principle of the aspect-oriented paradigm is well suited to addressing security concerns (Robinson et al., 2004). Aspect orientation has the potential to enhance the implementation of security concerns in terms of reusability and extensibility, thereby improving the robustness and maintainability of a system. Evidently, abstracting a security feature into a security aspect increases the possibility that it may be reused for other applications (Padayachee and Wakaba, 2007). Access control and encryption, for example, have similar requirements for most applications. Vanhaute and De Win (2001) demonstrated how to convert these security concerns into reusable generic aspects.

Several authors cite the benefits of using aspect-oriented programming for implementing security concerns (De Win, Vanhaute et al., 2002; Viega et al., 2001). According to Bodkin (2004), aspect-oriented programming is relevant for all major pillars of security: ‘authentication, access control, integrity, non-repudiation, as well as for supporting the administration and monitoring disciplines required for effective security’. Even security-related bugs such as buffer overflows or race conditions can be considered security-related concerns (De Win et al., 2003). Security aspects can be used to modularise access control and authentication (see (De Win et al., 2003); (Shah and Hill, 2003) and (Slowikowski and Zielinski, 2003)). The primary argument supporting aspect-oriented programming is that the average programmer does not have the requisite skills in security (Viega et al., 2001). This can be attributed to a lack of expertise and few tertiary institutions offering tuition in software security (Viega and Evans, 2000). Programming tasks such as authentication, access control and integrity should be abstracted away from developers and allocated to security experts. Secondly, it is observed that security concerns such as encryption and access control tend to crosscut the code base. Thirdly, a security aspect can be reused for other applications since access control has the same requirements for most applications (De Win et al., 2001). Fourthly, aspect-oriented software design is flexible enough to

accommodate the implementation of additional security features after the functional system has been developed.

This study proposes using the aspect-oriented paradigm to facilitate the non-intrusive insertion of access control features such as usage control into a fully operational software system. (This is validated by a proof-of-concept prototype that will be presented in Chapter 8.) Chapters 2, 3 and 4 explore traditional access control models, usage control and present the concept of optimistic access control respectively. Chapters 5 and 6 discuss the concepts of aspect-oriented programming and their relationship with security. Chapters 7 and 8 present the model itself and evaluates the model concept. Chapter 9 concludes the thesis by assessing the model presented and the implementation technique applied.

1.2 Motivation for this study

Discretionary access control is an access policy that restricts access to files and other system objects such as directories and devices based on the identity of the users and/or the groups to which they belong (Russell and Gangemi, 1991). In the case of discretionary access control, no control is enforced on the use or dissemination of the information once this information has been released to an authorised user (Pfleeger, 1997). For example, a subject *Jane* may at her own discretion decide whether *Sam* may read the file entitled *Logistics*, assuming she owns the file. Discretionary access control is very flexible but highly vulnerable to Trojan Horses. As a result of this inadequacy, mandatory access policies were proposed.

Mandatory access control (Ramachandran et al., 2006) refers to access control policy decisions that are made beyond the control of the individual owner of the object. A central authority determines what information is to be accessible by whom, and the user cannot change access rights (Pfleeger, 1997). With mandatory access policies, every object and user in the system is assigned a sensitivity label that consists of a level of secrecy and a set of compartments (Bell and La Padula, 1976). Mandatory access control is deemed to be superior to discretionary access control as it is not vulnerable to illegal information flows. An

illegal flow arises when information is transmitted from one object to another in violation of the information flow security policy (Samarati et al., 1997). Even the most dominant model of recent times, the role-based access control model, is vulnerable to illegal information flows, as is demonstrated by Chon et al. (2004). Within role-based access control (RBAC), system administrators create roles according to the job functions performed in a company or organisation, grant permissions (access authorisation) to those roles, and then assign users to the roles on the basis of their specific job responsibilities and qualifications (Sandhu et al., 1996).

These models often assume that users want and are able to determine permissions before the actual access is made. These mechanisms require a priori setting of permissions that are difficult to specify and maintain in highly dynamic environments. In this thesis, this category of access controls is referred to as *traditional access controls*. These types of access controls are characteristically pessimistic. In other words, the models assume that human beings cannot behave in a trustworthy manner and the system has to prevent them from behaving in an undesirable way. Human trust is subjective and context specific and hence it is difficult to form a definition that incorporates all views and types of trusts (Grandison, 2003). Integrating trust/distrust into the computing world requires transforming a complex social concept into an easy-to-use technical product that embodies the basic principles of trust/distrust (English et al., 2002). Human beings make decisions based on the circumstances of a particular situation. For example, within a typical mandatory access control model, *doctors* may have the privilege to view sensitive information but *nurses* and *clerks* would not. In the case of role-based access control, the role could be based on job responsibilities; for instance, a patient's record can be written by any health professional assigned to the role of ward physician (Pudney, 2003). However, this does not guarantee that a valid user demonstrates integrity or acts professionally.

Access controls are difficult to implement and are evidently deficient under certain conditions. Traditional access controls offer no protection for unclassified information – such as a telephone list of employees that is unrestricted, yet available only to members of the company. On the opposing side of the continuum, organisations such as hospitals that

manage highly sensitive information demand stricter access control measures. Yet, traditional access control may well have inadvertent consequences in such a context. Often, in unpredictable circumstances, users that are denied access could have prevented a calamity had they been allowed access. It has been proposed that controls such as auditing and accountability policies be enforced to deter rather than prevent unauthorised usage. In dynamic environments, preconfigured access control policies may change dramatically, depending on the context. Moreover, the costs of implementing and maintaining complex preconfigured access control policies sometimes far outweigh the benefits. This research considers an adaptation of usage control as a proactive means of deterrence control to protect information that cannot be adequately or reasonably protected by access control. Deterrent controls are intended to discourage individuals from intentionally violating information security policies or procedures. Hence, if software systems could trust humans to decide how and when they can access information, this would be a more accurate assessment of trust. Trust on a humanistic level is highly complex and there are a variety of factors that influence trust. The emergence of trust-based access control frameworks is largely due to communications occurring among parties where each party is unknown. This communication is typically decentralised. There is now a need for a new type of access control where the access is not preconfigured and where, essentially, the user is trusted to behave ethically.

While pessimistic access controls such as DAC, MAC and RBAC maybe highly appropriate in certain contexts, optimistic access controls may be more appropriate in other circumstances. For instance, Stevens and Wulf (2002) considered an actual inter-organisational co-operation scenario where it was found that traditional access control did not comply with the organisation's requirements and that co-operation and competitive reasons motivated the use of interactive and optimistic access controls. Hong and Landany (2004) also established that there is a need for privacy-sensitive systems to have a range of control and feedback mechanisms for building pessimistic, optimistic and mixed-initiative applications.

The approach of deterrence control is an application of optimistic access control. Optimistic access control is useful in cases where openness and availability are more important than complete confidentiality (Povey, 1999). Optimistic access control also has the advantage that it is far easier to implement, since it is rather difficult for administrators to predict all of the possible usage scenarios and thus all of the necessary permissions. Optimistic access control is based on the assumption that most access control processes will be legitimate, and relies on controls external to the system to ensure that the organisation's security policy is maintained. The scheme allows users to exceed their normal privileges in a way that is constrained, so that it is securely audited and may be rolled back (Povey, 1999). Optimistic access control involves a combination of audit and accountability; and deterrent mechanisms to encourage trustworthy behaviour. This approach is characteristically more retrospective rather proactive. However, the application of usage control within an optimistic access control context may provide a proactive means of deterrent control. Within traditional access control models, usage control would offer an extra layer of restriction against unauthorised usage. However, under the optimistic access control paradigm it would not restrict users but rather deter them from accessing and misusing information. As is defined in terms of the optimistic access control paradigm, the user must ultimately be able to access the requisite information.

Optimistic access controls trust human beings to perform legitimate accesses and take retrospective action once such trust has been breached. The initial cost of implementing optimistic access control methods is minimal; however, the fall-out could be disastrous. If such a breach is discovered, it could involve prosecution or performing a roll-back procedure. The roll-back procedure may be able to restore the system to its original state. However, it is highly likely that it may not be able to undo the damage done.

1.3 Problem Statement

Access controls are difficult to implement and maintain due to the highly complex task of envisaging all the possible usage scenarios and thus all the necessary permissions. Notwithstanding this fact, the pre-configured access controls may not be appropriate to all contexts. Furthermore, a security policy specified by a single access control model may not be applicable to all data in an information security system. This research considers an adaptation of usage control as a proactive means of deterrence control to protect information that cannot be adequately or reasonably protected by access control. This thesis therefore presents a model for reformulating usage control under the optimistic access control paradigm.

To accomplish the main goal identified above, the following sub-goals were identified:

- Providing a critical overview of access controls and optimistic access controls
- Providing an overview of aspect-oriented programming and its relevance to security
- Deriving a model that enforces **Optimistic Access Control with Usage Control** – designated the **OAC(UCON)** model – within a mixed-initiative access control framework
- Providing a proof-of-concept prototype to demonstrate the suitability of aspect-orientation in terms of implementing the model concept
- Providing evaluative prototypes of the model concept in a small-scale experiment using the design science methodology

1.4 Terminology used in this thesis

Access control is a fundamental part of computer security where every requested access must be governed by an access policy stating who is allowed access to what; i.e. the request must be mediated by an access policy enforcement agent (Pfleeger and Pfleeger, 2003).

Aspect-Oriented Programming provides explicit language support for modularising design decisions that cross-cut a functionally decomposed program (Walker et al., 1999), i.e. the developer is able to maintain the code (cross-cutting functionality) in a modularised form.

Mixed-initiative access control framework is an access control strategy that involves combining a minimum of two access control models where the request to information is mediated by a mixture of access policy enforcement agents.

Optimistic Access Control is a scheme that allows users to exceed their normal privileges in a way that is constrained, so that it is securely audited and may be rolled back.

Usage Control (UCON) is an access control model that encompasses emerging applications such as trust management in a unified framework.

1.5 Research Methodology

The research methodology involved the design of a proof-of-concept prototype to demonstrate a subset of the model concept and to evaluate the suitability of the aspect-oriented paradigm. Additionally, the model was evaluated in terms of the design science research method so as to test its scalability and efficacy as a security measure. As the value and utility of the model concept was evaluated, the design science method was selected to this end. It involves a two-step process of building and evaluating (March and Smith, 1995). During this process several evaluative prototypes were developed to verify the model concept for commercial systems. The small-scale experiment tested the theory that users' interaction with the prototype will be perceived as an effective countermeasure against data misuse. In order to test the hypothesis, two qualitative data collections were employed during the evaluation, namely participant observation and open-ended interviewing. Postgraduate students with an extensive knowledge of information systems were utilized to develop and evaluate the model concept. Since some of these students are already employed within the information systems sector, this profile of participants can often serve as representatives of systems developers.

1.6 Delimitations

It is important to distinguish between access control and information flow control. For example, an access policy might specify that user1 can read from file1 and write to file2, while a flow policy might specify that information in file1 is at most confidential and always less than the class of information in file2 (Andrews and Reitman, 1980). The present study is primarily concerned with access control, and more specifically with enhancing access control by means of usage control. Thus it will specifically consider the reformulation of usage control under the optimistic control access control paradigm.

Although the issue of trust is an important component of the mixed-initiative access control framework, it is beyond the scope of this thesis to provide details as to how trust can be measured and maintained. The issue of mutability of access rights based on trust is given due consideration – however, it is not explored in any detail.

1.7 Thesis Layout

Chapter 1: Introduction: The problem is introduced in relation to access controls in that they are too restrictive and difficult to pre-configure.

Chapter 2: Access Control: This chapter introduces traditional access control models and presents the problem with the traditional approach to access control.

Chapter 3: Optimistic Access Control: This chapter introduces optimistic access control and presents its strengths and weaknesses. While Chapter 2 covers the more traditional methods of access control, Chapter 3 focuses on a non-conventional method of implementing access control.

Chapter 4: Usage Control: This chapter introduces usage control and how it may be used to address the weaknesses of optimistic access control.

Chapter 5: Aspect-Oriented Programming: The aspect-oriented programming paradigm is introduced as a mechanism to implement access control measures. The abstraction of this chapter allows for an adequate overview of the aspect-oriented paradigm as it is a fairly new programming technique and not yet ubiquitous within the South African context.

Chapter 6: Aspect-Oriented Security: This chapter demonstrates how the aspect-oriented paradigm may be used within the information security domain.

Chapter 7: The OAC(UCON) model: This chapter presents the model that is used to address the inadequacies of the traditional access control requirements.

Chapter 8: Prototyping and Model Evaluation: This chapter describes the implementation of the model proposed as a "proof-of-concept" by using an aspect-oriented programming language. It also provides an evaluation of the approach and the model concept using evaluative prototyping.

Chapter 9: Conclusion: This chapter concludes with directions for future research and evaluates the contribution made by this thesis.

Figure 1.1 below presents a schematic overview of the thesis.

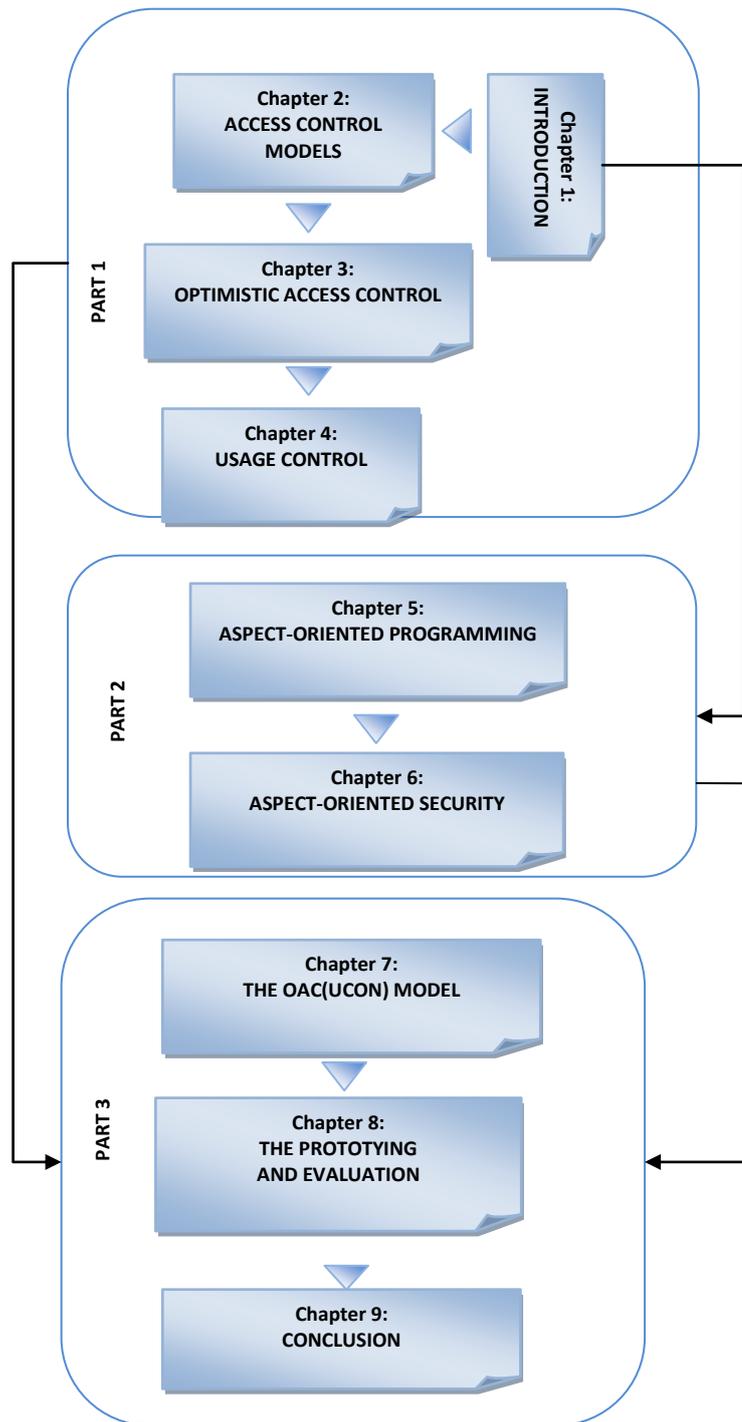


Figure 1.1: Overview of Thesis

1.8 Summary

The proposed solution offered by this study may ease the burden of system administrators significantly. It is rather difficult for administrators to predict all of the possible usage scenarios and thus all of the necessary permissions. With optimistic access control, it is ultimately left to users to make that judgement. The complexity of the implementation and maintenance of pre-configured access control policies is therefore relegated to the way the user interacts with the system. Adapting usage control as a deterrent mechanism provides a proactive mechanism that can be used in addition to the retroactive methods of auditing and accountability offered by optimistic access control. Through using a proactive means of deterrent control, a larger subset of information may be relegated into the public domain. This research does not obviate the need for traditional access control. For example, payment processing e-business applications demand stricter information controls (Haldar et al., 2005). Consequently the model presented here is intended to be incorporated into a mixed-initiative access control framework.

CHAPTER 2:

ACCESS CONTROL

2.1 Introduction

Access control is a fundamental part of computer security where every requested access must be governed by an access policy stating who is allowed access to what; the request must then be mediated by an access policy enforcement agent (Pfleeger and Pfleeger, 2003). There are three basic approaches to access control. The first approach requires pre-configured access control policies (explored in this chapter), while the second involves temporal access control policies (explored in Chapter 4). Chapter 2 provides an overview of traditional access control models such as discretionary access control (DAC), mandatory access control (MAC) and role-based access control (RBAC). The inadequacies of these models are explored, thus providing the rationale for investigating the third approach to access control – optimistic access control. This approach is characteristically retrospective.

2.2 Discretionary Access Control

Discretionary access control (DAC) is an access policy that restricts access to files and other system objects such as directories and devices based on the identity of the users and/or the groups to which they belong (Russell and Gangemi, 1991). With discretionary access control, no control is enforced on the use or dissemination of the information once this information has been released to an authorised user (Pfleeger, 1997). For example, a subject *Jane* may at her own discretion decide whether *Sam* may read the file entitled *Logistics*, assuming she owns the file (see Figure 2-1).

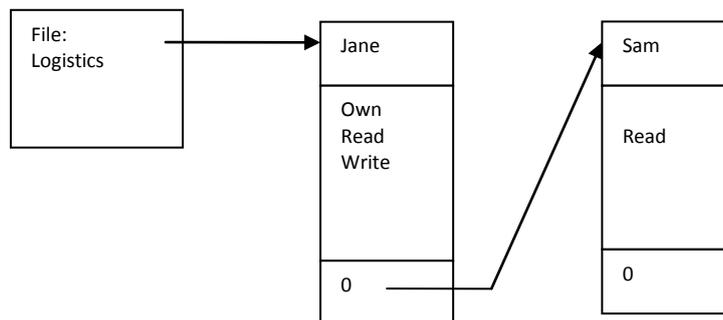


Figure 2-1: Discretionary Access Control based on an Access Control List (adapted from(Tolone et al., 2005))

According to Pieprzyk et al. (2003), there are several deficiencies with this type of enforcement:

- If a permission α is transferred from one subject to another, then the second subject can propagate the permission α with no agreement from the first subject.
- The read permission allows a reader to copy the object and to grant friendly subjects read access to the copy.
- If two or more untrustworthy processes conspire, they may exercise their permissions collectively.

Discretionary access control is particularly flexible, yet highly vulnerable to Trojan Horses (Downs et al., 1985). According to Li et al. (2009), this is because it is assumed that all programs are benign and will not be exploited by malicious inputs. McCollum (1990) contends that discretionary access control is inappropriate for the enforcement of an 'integrated, global access policy based on a comparison of explicit markings on data to attributes of the user seeking access', as such controls are designed to relate individual users to specific data objects. (Li et al., 2009) go on to add that the problem with discretionary access controls is that the enforcement mechanism cannot correctly identify the true origins of a request made by multiple principles. To this end, there has been an inclination to complement DAC mechanisms with some form of mandatory access control (Mao et al., (2009).

2.3 Mandatory Access Control

Mandatory access control refers to access control policy decisions that are made beyond the control of the individual owner of the object. A central authority determines what information is to be accessible by whom, and the user cannot change their access rights (Pfleeger, 1997) (see Figure 2-2). With mandatory access policies, every object and user in the system is assigned a sensitivity label that consists of a level of secrecy and a set of compartments (Bell and La Padula, 1976). For example, as depicted in Figure 2-2 below, the sensitivity label of the Logistics file is SECRET [ALPHA, VENUS], where SECRET indicates the level and [ALPHA, VENUS] the compartments. The security level is an element of a totally ordered set. The levels generally considered are: TOPSECRET, SECRET, CONFIDENTIAL and UNCLASSIFIED, where TOPSECRET > SECRET > CONFIDENTIAL > UNCLASSIFIED (Russell and Gangemi, 1991). The set of compartments is unordered. An access class c_i dominates (\geq) an access class c_j if and only if the security level of c_i is greater than or equal to c_j and the compartments of c_i include that of c_j . Access control is based on the following two principles formulated by Bell and LaPadula (1976), which are adopted by all models enforcing mandatory access security policies:

- No read-up: A subject can read only those objects whose access class is dominated by the access class of the subject, namely the Simple Security Property.
- No write-down: A subject can write only to those objects whose access class dominates the access class of the subject, namely the *-Property.

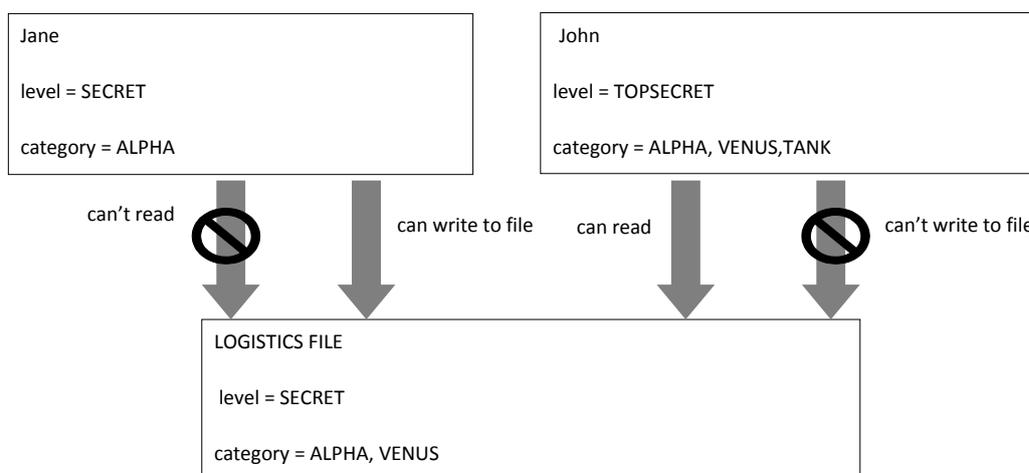


Figure 2-2: Mandatory access control (MAC) (adapted from (Russell and Gangemi, 1991))

It is important to note that the term '*object*' does not imply 'object' in the typical object-oriented sense. In fact, the term 'subject' is the active process that requests access to the 'object', which are passive entities such as files or records.

Although mandatory access control is considered to be superior to discretionary access control, it is difficult to implement in reality and the model has a number of deficiencies as indicated below (Anderson, 2001):

- 'Blind write-up' – The inability to inform low-security data whether a write to high-security data has happened correctly
- 'Downgrading'– Moving information from a high-security level to a lower level is sometimes desirable
- 'TCB bloat' – A large subset of the operating system may end up in the Trusted Computing Base (TCB)

Mandatory access control (MAC) was once thought to be relevant to the military only. These days, however, it is gradually being incorporated into commodity open operating systems such as BSD and Linux. As Zakrzewski and Haddad (2002) put it, 'mandatory access control mechanisms are efficient for supporting complex relationships between different entities in the computing environment'. Systems such as payment processing, e-business applications and medical data applications may also require similar stringent controls. Mandatory access control is highly applicable in areas such as privacy, as access to privacy-sensitive data can be regarded as analogous to access to multilevel security data (Rjaibi and Bird, 2004). Maintaining the privacy of individuals is one of the most compelling reasons for implementing strong access controls in an organisation. Weippl and Essaymr (2003) also demonstrate that besides its applicability to the military, mandatory access control has the efficacy to protect personal digital assistants (PDAs). Another reason why mandatory access control is deemed to be superior to discretionary access control, is because it is not vulnerable to illegal information flows. An illegal flow arises when information is transmitted from one object to another object in violation of the information flow security policy (Samarati et al., 1997). Even the most dominant model of recent times, the role-based

access control model, is vulnerable to illegal information flows, as demonstrated by Chon et al. (2004).

Mandatory access control can be easily unified within the role-based access framework, since role-based access control (Sandhu, 2001) is a means of articulating policy rather than embodying a particular security policy (Osborn et al., 2000). Mandatory access controls generally cannot prevent implicit flows arising from the control paths not taken at run time (Zheng and Myers, 2004). To prevent an information leak like this, Denning and Denning (1977) proposed a mechanism to certify that a program does not violate information flow policy. Information flow is concerned with the control path of information as a software system executes. There exists a semantic gap between access controls of operating systems and programming languages as languages such as the Java Virtual Machine lack mechanisms to enforce mandatory access controls, programming languages have been created (Haldar et al., 2005).

2.4 Role-based Access Control

Within role-based access control (RBAC), system administrators create roles according to the job functions performed in a company or organisation. They grant permissions (access authorisation) to those roles, and then assign users to the roles on the basis of their specific job responsibilities and qualifications (Sandhu et al., 1996).

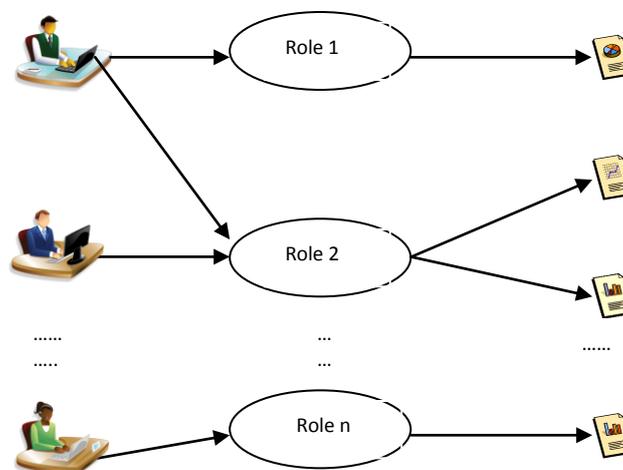


Figure 2-3: Role-based Access Control (adapted from (Samarati and de Capitani di Vimercati, 2001))

According to Tolone et al.(2005), the shortcomings of role-based access control can be summarised as follows:

- The nature of the roles is static and they lack flexibility and responsiveness to the environment in which they are being used.
- It lacks the ability to specify fine-grained control of individual users in certain roles and individual object instances.
- Constraints are an important aspect of role-based access control and a powerful mechanism for stipulating high-level organisational policy, the specification of which is not expanded on in the model.

Izaki et al. (2001) demonstrate how illegal information flow may occur among objects within the role-based access control model. As with discretionary access controls, role-based access control can only restrict the access of objects in a system – hence information flows among variables cannot be controlled (Chou, 2003).

Role-based access control does not scale up with access control model issues. The core is for the most part unchanged and based largely on the access matrix model (Zhao et al., 2007). This inflexibility is to be addressed by the model concept presented in this thesis. The aim is to provide a model that is not entirely dependent on subject-object attributes (as with the models presented in this chapter), but rather a flexible model that is dependent on temporal factors.

2.5 Conclusion

Traditional access control is based on static authorisations that depend exclusively on a subject's permissions with regard to target objects (Zhao et al., 2007). Usage control seeks to address these inadequacies as it considers other factors that may influence a subject's rights to a target object. Moreover, it considers the mutability of attributes during access. Usage control will be explored in more detail in Chapter 4. Traditional access control models are based entirely on denial of access. They do not consider contextual factors or extenuating circumstances that may warrant overriding these controls. Access control

models such as DAC, MAC and RBAC often assume what users want and are able to determine permissions before the actual access is made. They require permissions to be pre-configured, which is difficult to specify and maintain in highly dynamic environments where access policies may fluctuate on a regular basis. In the next chapter optimistic access control is considered. It is based on the assumption that most accesses will be legitimate and the control is retrospective.

CHAPTER 3:

OPTIMISTIC ACCESS CONTROL

3.1 Introduction

Industry surveys prove that a substantial share of computer security incidents are due to the intentional actions of legitimate users. – the consequences of which include negative publicity, competitive disadvantage and loss of consumer confidence (D' Arcy and Hovav, 2007). Traditional access control models are evidently deficient under certain conditions. For instance, a particular organisation may necessitate access controls to be less prescriptive for the purposes of intra-organisational cooperation (Etalle and Winsborough, 2007; Stevens and Wulf, 2002). Traditional access controls such as mandatory, discretionary or role-based access control offer no protection for information that is unclassified and freely available in the public domain. In the recruitment industry, for example, information such as client lists and candidate lists has to be shared freely for the purposes of collaborative job matching. As there are no controls over this information, an employee may well download it and distribute it to competitors.

On the opposing side of the continuum, organisations (e.g. hospitals) that manage highly sensitive information stipulate stricter access control measures. Yet traditional access controls may sometimes have an undesired effect in these circumstances as well, for instance the denial of access based on the attributes of the users rather than the context of the access. The meaningful implementation of access control remains a difficult task and preconfigured access control policies may at times change dramatically in dynamic environments, depending on the context. Moreover, the costs of implementing and maintaining complex preconfigured access control policies sometimes far outweigh their benefits. It has been proposed that auditing and accountability measures be enforced to

deter unauthorised users rather than to completely prevent them from gaining access (Etalle and Winsborough, 2007). While pessimistic access controls such as DAC, MAC and RBAC may be highly appropriate in certain contexts, optimistic access controls may be more appropriate in other circumstances. This issue is investigated in Section 3.4.

3.2 Optimistic Access Control

Optimistic access control is useful in cases where openness and availability are more important than complete confidentiality. Optimistic access control also has the advantage that it is far easier to implement, since it is difficult for database administrators to predict all of the possible usage scenarios and thus all of the necessary permissions. Optimistic access control is based on the assumption that most access control processes will be legitimate, and relies on controls external to the system to ensure that the organisation's security policy is maintained. The scheme allows users to exceed their normal privileges in a way which is constrained, so that it is securely audited and may be rolled back (Povey, 1999).

According to Povey (1999) the optimistic enforcement of security policies are retrospective and rely on administrators to detect unreasonable access and take steps to compensate for the action. Such steps might include:

- Undoing illegitimate modifications
- Taking punitive action (e.g. firing or prosecuting individuals)
- Removing privileges

Optimistic access controls trust human beings to perform legitimate accesses and take retrospective action after such trust has been breached. This approach is characteristically more retrospective rather proactive. However, the application of usage control within an optimistic access control context may provide a proactive means of deterrent control. Povey (1999) suggests using integrity to complement optimistic access control, where the user is unable to manipulate data arbitrarily. However, this thesis is not concerned with maintaining the integrity of the data – rather, the model presented here involves protecting the access of information.

3.3 Requirements for Optimistic Security

Since the seminal article on optimistic access control was written by Povey (1999), the next three sections are based largely on his work. Providing an optimistic security system requires mechanisms to ensure that the likelihood and consequences of a user maliciously using or plainly misusing the system are minimised. In order to satisfy this objective, the following controls should be considered:

- **Constrained entry points:** Users should not in general be allowed to exceed their privileges. Users should be warned when they exceed their privileges and be reminded of their obligations towards their organisation.
- **Accountability:** The system must have strong enforcement of authentication so that users are associated with their actions.
- **Audit:** The system must log the actions of users in detail, so that a post-mortem analysis can determine whether an access has been legitimate or not.
- **Recoverability:** There should a mechanism for the system to be rolled back to ensure that a user cannot damage a system irreparably.
- **Deterrents:** One effective way of reducing risks in an optimistic system is by using punitive measures to deter misuse. The punitive measures themselves can be either optimistic (with the system administrator enforcing the measure on the detection of misuse) or pessimistic (with the punitive measure implemented immediately and reversed if the action is determined to be legitimate).

In the model presented in Chapter 7, the constrained entry points stipulation is satisfied by pre-obligations and pre-conditions offered by usage control. The accountability notion is addressed by the fact that only authorised users are allowed access. The model system retrospectively provides mechanisms for audit and recoverability. The issue of deterrents is enforced by the obligations and conditions offered by usage control.

3.4 Applicability of optimistic security

According to Povey (1999), optimistic access control may be applied in the following contexts:

Emergency "break-glass" tool

The software equivalent of the "break-glass" container would be a program that is suitably constrained using an optimistic security system and that gives stern warnings about misuse before it is activated. This mechanism is incorporated in the model presented in Chapter 7.

Retrospective content filtering

One of the negative aspects of systems that provide filtering of material which is deemed harmful or inappropriate is that the algorithms used to determine which content to filter can often result in false matches. The result of this is that users can be denied access to legitimate content, forcing them to search for ways to circumvent the system. By applying the principles of optimistic security, users would be able to access any material they desired, and an administrator would log all material accessed and run the content-filtering algorithm retrospectively.

Sandboxing "somewhat-trusted" applications

Traditionally, the focus of "sandboxing" (or constraining the access privileges of programs) has been on untrusted code that is downloaded from the Internet. For example: an optimistic sandbox could track the changes made to the file system by a word-processing program, and allow the user to undo these changes in the event of a crash or malicious macro virus. This would improve the security and safety of these applications without the loss of functionality or expensive certification of the programs.

Watching your system administrator

The system administrator can be constrained in that the user is informed whenever the administrator accesses files that may involve a breach of the user's privacy.

With regard to the context of optimistic access control, it relies entirely on the user being accountable for its own compliance to access control policies, rather the system enforcing access control policies on the user and controlling the user's action. In terms of the compliance mindset subscribes to what might be called a deterrence theory of motivation, which employs mandates, procedural controls and threats of punishment to manage and motivate people (Herath and Rao, 2009). Deterrence theory is based on certainty, severity and celerity of punishment that affect people's decisions on whether or not to commit a crime or not (Higgins et al., 2005). In an information systems security context, these may be visualised in terms of an employee's assessment of the consequences of a security threat and the probability of exposure to a substantial security threat (Herath and Rao, 2009).

3.5 The extensibility of the Optimistic Access Control Model

Optimistic access controls address this niche where access control is not preconfigured and the user is essentially trusted to behave ethically. While traditional access controls such as DAC, MAC and RBAC may be highly appropriate in certain contexts, optimistic access controls may be more appropriate in other circumstances. A field study conducted by Stevens and Wulf (2002) who considered the cooperation between two engineering offices and a steel mill is a case in point. Within this real-world inter-organisational co-operation scenario, it was found that traditional access controls did not comply with the organisation's requirements and that co-operation and competitive reasons motivate the use of interactive and optimistic access controls (Stevens and Wulf, 2002).

A posterior policy enforcement offers interoperability, flexibility and scalability, which are crucial in collaborative environments (Etalle and Winsborough, 2007). Cederquist et al. (2006) also considered enforcing usage control policies a-posterior. This notion is similar to the optimistic access control. The ideas expressed by Cederquist et al.(2006) are different from the model presented in Chapter 7, as they focused on the auditing aspect of enforcing and updating usage control policies retrospectively. The OAC(UCON) model uses usage control as a deterrent mechanism to proactively prevent users from committing data misuse. Thus the idea of refining a policy should be the exception rather than the norm.

Additionally, the OAC(UCON) model focuses more on the pragmatic level of implementation at the application level. The OAC(UCON) model includes the optimistic access control requirements of roll-backs and the issue of continuity in terms of usage control. The model also considers how to leverage optimistic access control within the wider context of traditional access control. The issue of how access control policies will be enforced after a threat has been discovered are beyond the scope of this research. Typically such enforcement will be audit-based (Cederquist et al., 2006). However, a neural network or an expert system could also be applied to decrease a user's privilege to information in the public domain. This approach may prove to be more effective as the updates to access control policies could be more synchronous. This matter will be expanded on in Chapter 9 as a direction for future research.

The mixed-initiative approach is also gaining recognition, where traditional access control is combined with optimistic access control approaches. Imine et al. (2009) consider the domain of distributed collaborative editors that provide support for modifying simultaneously shared documents – such as sharing programming code among dispersed users. Controlling access in such systems is challenging due to dynamic access changes and low latency access to shared documents. They complemented mandatory access control with optimistic access control to solve this problem. To deal with the latency and dynamic access changes, optimistic access control was applied where the enforcement was retrospective. Briscoe et al. (2000) presented a multi-service packet network which involved a mixed-initiative approach where optimistic access control existed within the wider context of the pessimistic access control. For example, the customer may be given an Internet account after providing verifiable identification. Once past this pessimistic hurdle, the optimistic access to more specific parts of the system can be allowed, yet is enforced by punishment. The rationality for such enforcement is that it leaves the network structure clear to simply classify, route, schedule and forward.

The call for privacy-sensitive systems to have a range of control and feedback mechanisms for building pessimistic, optimistic and mixed-initiative applications has also been recognised (Hong and Landay, 2004). Esquivel et al. (2007) also employed optimistic access

control in terms of privacy in pervasive environments. There are environments capable of sensing personal information anywhere and at any time. Based on the “fair-trade” metaphor, they presented a privacy solution dealing with a user’s privacy as a tradable good for obtaining services in an environment. Thus, users gain access to more valuable services as they share more personal information. This strategy combined with optimistic access control and logging mechanisms, enhances user confidence. Zhao and Johnson (2008) propose access governance with both flexibility and security of information systems. They combine information access, audit, violation penalties and rewards so as to enable self-interested employees to access information in a timely manner and seize business opportunities for the organisation, while managing security risks.

While protecting information from misuse, managers strive to ensure that employees can actually access the information they need to create value (Zhao and Johnson). In terms of the models presented above, optimistic access control is applied retrospectively, whereas the OAC(UCON) model, a flexible system with usage control to prevent misuse, is applied proactively.

The flexibility of the optimistic access control paradigm may be used to address circumstances where traditional access controls prove to be inadequate. However, optimistic access control must be qualified with enforcements to prevent data abuse. In terms of the OAC(UCON) model, it formalises the requirements for optimistic security such as constrained entry points, accountability, audit, recoverability and deterrents. Furthermore, the extensibility offered by this paradigm is complementary to a mixed-initiative access control framework into which the OAC(UCON) model is intended to be incorporated.

3.6 Conclusion

The implementation of optimistic access control requires minimal effort and is cost efficient, because it does not involve specifying and maintaining access control rights. However, the flexibility offered by optimistic access control is a security risk and subject to misuse. Hence it is proposed that this type of access control should be augmented with some sort of control to ensure that humans behave ethically. It is proposed that optimistic access control be complemented with usage control. Within traditional access control models, usage control would offer an extra layer of restriction from unauthorised usage. However, under the optimistic access control paradigm it would not restrict users, but rather deter and constrain them from accessing and misusing information. As was stated earlier, the user should ultimately be able to access the required information.

CHAPTER 4:

USAGE CONTROL

4.1 Introduction

Sandhu and Park (2003), recognising the inadequacy of traditional access control models, proposed a new approach to access control called usage control (UCON). The usage control model is highly appropriate in dynamic and distributed environments where other decision factors such as context should be included to offer stricter enforcement on the rights to digital objects. Typically, access controls consider enforcements that are made prior to access; the UCON model extends pre-authorisation by re-evaluating usage requirements throughout usages. This property is called “continuity” and has to be captured in modern access control for the control of relatively long-lived usage or for immediate revocation of usage (Sandhu and Park, 2003). The other unique property of the UCON model is "attribute mutability". In modern information systems, the decision policies may change as a consequence of certain actions that may result in modifications to the object or subject attributes (Park et al., 2004). The UCON model has been largely inspired from digital rights management and is a general purpose, unified framework that encompasses traditional access control, trust management and digital rights management (Park et al., 2004). This chapter provides an overview of the UCON model and elaborates on its applicability and extensibility.

4.2 The continuity and mutability of the UCON model

The UCON model encompasses emerging applications such as trust management in a unified framework. It is claimed that the missing components of traditional access control are the concepts of *obligations* and *conditions*. *Obligations* require some action by the subject so as to gain or sustain access, e.g. by clicking the ACCEPT button on a licence agreement or agreeing not to distribute the document. *Conditions* represent system-oriented factors such as time of day, where subjects are allowed access only within a specific time period.

In addition to these three decision factors (namely conditions, obligations and authorisations) decision factors, there are two important properties called *continuity* and *mutability*. Continuity is useful when there is sustained usage over a long period of time. Hence usage requirements would be evaluated throughout the usage – known as ongoing authorisation. The other property is mutability, which is useful in Digital Rights Management systems where attributes have to be updated as a side-effect of a subject's actions. These updates may be before (pre), during (ongoing) or after (post) usages (see Figure 4-1). Typically, attribute management can be either admin-controlled or system-controlled. The admin-controlled system is immutable in that the attribute modification is at the administrator's discretion, whereas mutable attributes are automatically modified by the system at the time of usage (Park et al., 2004).

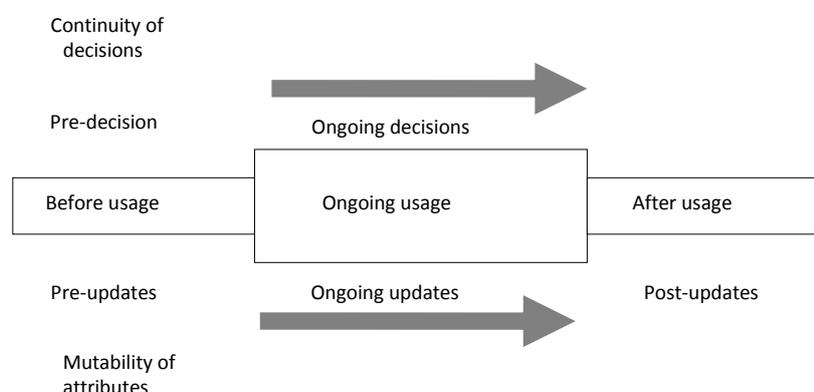


Figure 4- 1: Continuity and Mutability Properties (Park et al.).

4.3 The ABC Model for Usage Control (UCON model)

Sandhu and Park (2003) have expanded usage control into a family of models for usage control involving pre-authorisations and ongoing-authorisations. The implementation of pre-authorisation is relatively simple as it warrants checking the conditions and obligations before the user may proceed. However, the implementation of ongoing authorisation is non-trivial. Sandhu and Park (2003) do not offer a proposition towards how ongoing authorisations may be implemented.

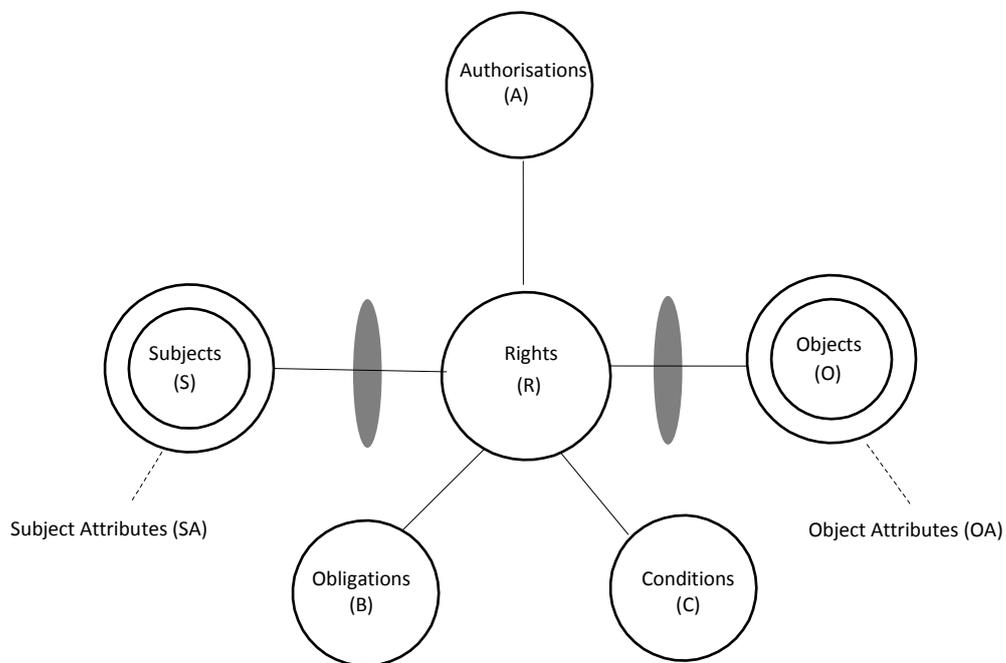


Figure 4-2: ABC Model Components(Sandhu and Park, 2003)

The ABC module (Figure 4-2) consists of eight components: subjects, subject attributes, objects, object attributes, rights, authorisations, obligations and conditions.

Each of these terms (adapted from Sandhu and Park (2003) and Park et al. (2004)) is explained briefly below:

- **Subjects:** represent users.
- **Objects:** target resources in respect of which subjects hold rights.
- **Right:** enables access of a subject to an object in a particular mode, such as a read or write access.
- **Subject and Object attributes:** properties that can be used during the access decision process. In practice, one of the most important subject attributes is subject identity.
- **Authorisation:** based on subject and object attributes.
- **Obligations:** requirements that a subject must perform before (pre), after (post) or during (ongoing) access.
- **Conditions:** environmental or system-oriented factors.

In terms of traditional access control, authorisation is assumed to be done before access is allowed (pre). However, the UCON model extends this to continuous enforcement by re-evaluating usage requirements throughout usages (ongoing). This type of enforcement, known as 'continuity', implies that access may be revoked instantaneously. Ongoing authorisation is active throughout the usage of the requested right and is repeatedly checked for sustaining access. Technically, these checks are performed periodically, based on time or events. For instance, suppose an ongoing obligation condition stipulates that a window declaring the *'Terms and Conditions of Use'* remains open during access. Thus, if the user ignores this stipulation and closes the window during access, the usage is revoked immediately (Sandhu and Park, 2003).

Usage control is relevant in many contexts, including privacy, digital rights management, management of Internet protocols and protocols for trade and administration secrets (Pretschner and Walter, 2008). A key motivation for using usage control is that it considers ongoing controls for extended access or for revocation. For example, Zhang and Nakae (2006) employed the UCON model for collaborative systems by leveraging the features of decision continuity and attribute mutability. This was ratified by the notion that traditional access control approaches do not consider the usage status of a shared object in authorisation. They developed a prototype and found that the main overhead of the system introduced by usage control involved mutable attribute acquisitions; policy interpretations and evaluations; and the updates of mutable attributes. Wang et al. (2006) also motivated using the UCON model for extended access, as it would be useful in ubiquitous environments where the information can be accessed anywhere and at any time, which is potentially unsafe. The ongoing continuity for authorisations, obligations and conditions found in the UCON model can be used to control objects in a dynamic environment since they provide more robust access control for ubiquitous computing environments and can protect sensitive messages from being disseminated.

4.4 The Usage Control Model architecture

From an architectural point of view, one of the most critical issues in enforcing the UCON model is the reference monitor. The reference monitor (Figure 4-3) associates decision policies and rules for the control of access to digital objects.

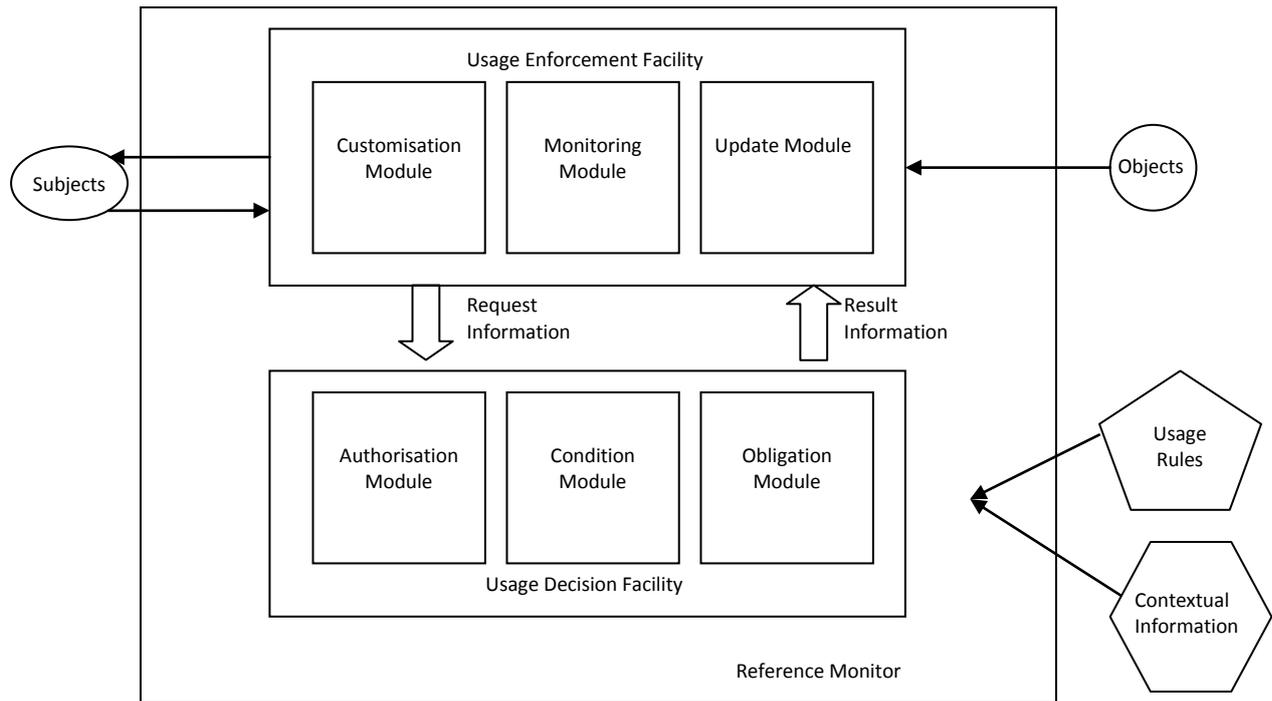


Figure 4-3: Conceptual Structure for the UCON Reference Monitor (Sandhu and Park, 2003)

Usage decisions are based on subject attributes, object attributes, authorisations, obligations and conditions at the time of the usage requests. The UCON reference monitor consists of a **Usage Decision Facility** and a **Usage Enforcement Facility**. The Usage Decision Facility includes modules for controlling the conditions, obligations and authorisations:

- The Authorisation Module deploys a process similar to traditional access control and utilises subject information, object information and usage rules to check whether the request is allowed or not.
- The Authorisation Module may return metadata to the Customisation Model indicating how the data may be presented or customised.
- The Condition Module uses the contextual information and usage rules to decide whether the conditional requirements have been met.
- The Obligation Module decides whether certain obligations need to be performed before, after or during access.
- If there is an obligation that must be performed, this is monitored by the Monitoring Module.
- The result will be resolved by the Update Module, which may change the attributes of the subject and/or object.

4.5 The Applicability and Extensibility of the UCON model

The UCON model is unique in that it can be applied in several contexts with differing access control policy strategies. The model encompasses other temporal and contextual factors aside from considering access rights on subject-object attributes. Consequently, there has been a trend towards complementing access control methods such as role-based access control with usage control (see Li et al. (2005) and Xu et al. (2003)). As indicated, usage control is relevant in many areas, including in privacy and digital rights management.

Although the UCON model is comprehensive, it has been extended in several ways. For instance, according to Lee et al. (2004) this framework lacks an important component in terms of access control. They maintain that the element of '*consent*' should also be included in an access control system, thereby increasing society's trust of a software system. In this scenario, consent is considered to be diametric to the 'concept of obligation' within the usage control model. 'While the obligation is obeyed by the customer, *consent* is observed by the provider' (Lee et al., 2004). The proposed method can extend the coverage of the UCON model in terms of security and enhance the right of both provider and customer. It also provides a solution for trust relationships in e-Commerce and for the protection of an individual's privacy. In a position paper, Pretschner and Walter (2008) considered usage control in the context of distributed systems that are composed of different actors assuming the role of data providers (who give data away) and data consumers (who request and receive data). In their position paper, they considered the element of negotiation for usage control. The term negotiation suggests that multi-step bidirectional communication takes place. Shin and Yoo (2007) extended the UCON model by incorporating an additional component, *delegation*, for effective modelling of the delegation of access rights in ubiquitous computing.

Syalim et al. (2005) proposed an enforcement to support data confidentiality in a database service provider by contextualising the usage control model and architecture for the aforementioned database service provider. In this context, the UCON model extended the access matrix by utilising either a server-side or client-side reference monitor, or both server-side and client-side reference monitors. Syalim et al. (2005) exploited the flexibility

of the UCON and separated the control domain in a database service provider into two parts: a database provider domain and a database user domain. In the database provider domain, the access control system controls users' access to the database services, while in the database user domain, the access control system controls other users' access to a user's database.

Due to the dynamic authorisation requirements in collaborative systems, Zhang and Nakae (2006) proposed a generalised authorisation framework for such systems based on the UCON model. In collaborative systems, organisations share their computing resources to establish virtual organisations. By leveraging the flexible policy specifications and attribute mutability of the UCON model, Zhang and Nakae's (2006) model supports virtual organisations-level authorisation policies, but also usage constraints defined by each resource provider. In their generalised authorisation framework, conditions are used to support context-based authorisations in ad hoc collaborations.

Due to the extensibility and expressibility of the UCON model and its all-encompassing nature, it may be extended to optimistic access control as well. The current research considers adapting usage control as a proactive means of deterrence control to protect information that cannot be protected adequately or reasonably by access control.

4.6 Conclusion

It has been suggested that where usage control on data consumers may not always be practical, feasible or sensible, optimistic access control may be more appropriate where the 'observation' or monitoring of user behaviour could be used as a deterrent (Pretschner et al., 2008). As discussed in Section 4.5, the UCON model is applicable in many areas due to its strong expressive power and policy specification flexibility (Zhang et al., 2006). The application of the UCON model to optimistic access control will be explored in detail in Chapter 7.

Usage control involves pre-authorisations and ongoing authorisations. The implementation of pre-authorisation is relatively simple, as it warrants checking the conditions and obligations before the user may proceed. In contrast, the implementation of ongoing authorisation is non-trivial, and Sandhu and Park (2003) have not proposed any stipulations as to how ongoing authorisations may be implemented. It has in the meantime been suggested by Padayachee and Eloff (2007) that multithreading should be applied to implement ongoing authorisations. However, in general, there is no notion of how usage control as a whole may be implemented in the real-world context to complement existing access control approaches. The aspect-oriented paradigm is now considered to be suitable for this context as it facilitates the abstraction of usage control decisions from other access control mechanisms and application logic. This approach would result in an implementation that is easier to augment to existing access control implementations. Before the OAC(UCON) model and the prototype are presented in Chapters 7 and 8 respectively, Chapter 5 unpacks aspect-oriented programming and the evolution that has led to this new programming paradigm.



PART 2:

**CHAPTER 5:
ASPECT-ORIENTED PROGRAMMING**

**CHAPTER 6:
ASPECT-ORIENTED SECURITY**

CHAPTER 5:

ASPECT-ORIENTED PROGRAMMING

5.1 Introduction

Aspect-oriented programming was first proposed by Gregor Kiczales and others at Xerox PARC in 1997. They developed it to offset redundancy in programs, thereby reducing complexity. Redundancies usually become apparent in areas such as security, memory management, resource sharing, and error and failure handling (Miller, 2001). Using aspect-oriented programming to address these redundancies is beneficial, as it has the potential to improve the reliability, maintainability, reusability (Viega and Voas, 2000) and robustness of an application. Although aspect-oriented programming is not ubiquitous in industry, it is receiving considerable attention from research and practitioner communities such as IBM, Northeastern University in the United States, the University of Twente in the Netherlands and Xerox (Miller, 2001). More recently, aspect-oriented software development has been successfully applied by Motorola, Siemens and Hewlett-Packard (Pohl et al., 2008). The field has matured to such an extent that it generated its own conference, and the first International Conference on Aspect-Oriented Programming took place in Twente in the Netherlands in 2002.

The object-orientated paradigm was found to be inadequate in terms of design and the implementation of cross-cutting concerns, as there is no elegant way of modularising such concerns. Aspect-oriented programming provides explicit language support for modularising design decisions that cross-cut a functionally decomposed program (Walker et al., 1999). That is, the developer is able to maintain the code (cross-cutting functionality) in a

modularised form. It is important to note that aspect-orientation maintains all the benefits of the object-oriented paradigm and should be viewed as an extension rather than a replacement of object-oriented technologies.

This chapter will commence with a discussion on the evolution of the different programming paradigms, after which the concept of aspect-oriented programming will be introduced. The next elaboration will cover the difficulties to be encountered with regard to the integration of aspect-oriented programming into software development.

5.2 Evolution to Aspect-Oriented Programming

Programming languages have evolved from assembly languages in the 1950s, to procedure-oriented languages in the 1960s, followed by structured programming and data abstraction in the 1970s. The next innovation involved object-oriented, distributed functional, and relational paradigms in the 1980s (Wegner, 1990). The sections that follow briefly describe the evolution of the aspect-oriented paradigm.

Structured programming enabled developers to provide modularity and reusability of constructs like procedures and functions (Constantinides and Hasson, 2002). However, as the size of software products increased, the structured programming paradigm was found to be inadequate and maintenance was becoming increasingly problematic. The basic tenets of the object-oriented programming paradigm, which embraced encapsulation and reuse, were viewed as a means for improving the quality and maintainability of software. The other advantage that the object-oriented paradigm had over the structured paradigm was the notion that it promoted thinking about software in a way that closely modelled the way humans perceive and interact with the real world. However, the object-oriented paradigm has not yet solved the problem of the duplication of code scattered throughout different objects. With object-oriented programming the attempt to reduce duplication was class inheritance. However, the problem is not related to the concept of inheritance. Instead, it is about unrelated objects that share some points of commonality (Padayachee and Eloff, 2006). The design and implementation of cross-cutting concerns still pose a problem to

object-oriented programming, as there is no elegant way of modularising these concerns. Aspect-oriented programming (AOP) addresses this problem and provides explicit language support for modularising design decisions that cross-cut a functionally decomposed program (Walker et al., 1999). In other words, the developer is able to maintain the code (cross-cutting functionality) in a modularised form.

Aspect-oriented programming is designed to exploit some of the advantages of object-oriented programming such as functionality, encapsulation, hierarchical classes and modularity. At the same time it manages to overcome an important disadvantage: objects cannot solve the problem of concerns that are not confined to a single class (Miller, 2001). These concerns result in the ‘tangling-of-aspects phenomenon’ (Kiczales, 1996), which inevitably results in a system that is difficult to maintain (Raje et al., 2001). Aspect-oriented programming advocates abstracting these cross-cutting concerns into modular units called aspects. For instance, suppose a system had one cross-cutting concern, scattered throughout the system. Suppose also that this cross-cutting concern can be removed from the system and codified as a single separate aspect. Thereafter this aspect is woven into the system at specific points when required, thereby simplifying the code significantly and also promoting more effective coding of these aspects. Furthermore, an update can be done on the single entity or aspect, rather than searching across the whole system and modifying the cross-cutting concern several times (Padayachee and Eloff, 2006).

As with aspect-oriented programming, subject-oriented programming is also based on the separation-of-concerns principle. In the subject-oriented paradigm, applications are composed of subjects. The subject-oriented paradigm defines the state and behaviour pertinent to the application itself – usually as fragments of the state and behaviour of collections of relevant classes (Harrison and Ossher, 1993). Subject-oriented programming also addresses some of the object-oriented programming limitations such as non-invasive system extensions and system decomposition. The main difference between aspect-oriented programming and subject-oriented programming is that aspect-oriented programming achieves non-invasive system extension through intercepting base code at

join points, whereas subject-oriented programming achieves system extension through the development of composition rules needed to integrate existing components.

5.3 Aspect-Oriented Programming Terminology

Aspect-oriented languages have three critical elements: a join point model, a means of identifying join points and a means of affecting implementation at these join points (Kiczales et al., 2001). There are two categories of aspects: development and production aspects. Development aspects can be used during the development of an application to facilitate debugging, testing and performance tuning, and they are not part of the final build of the application. Production aspects are intended to be included in the build of an application and they provide additional functionality for the application.

Here follows brief definitions of terminology used in aspect-oriented programming extracted from Kiczales et al. (2001) and Kiczales et al. (1997):

Cross-cutting

Behaviour that cannot be encapsulated. Because of its impact across the whole system, it is called cross-cutting behaviour.

Join Points

Join points are certain well-defined points in the execution flow of a program.

Pointcut

A pointcut is a set of join points described by a pointcut expression. The pointcut is used to find a set of join points where aspect code would be inserted.

Advices

Advice declarations are used to define additional code that runs at join points. For example, AspectJ supports before and after advice, depending on the time the code is executed. 'Before' (after) advice on a method execution defines code to be run before (after) the

particular method is actually executed. ‘Around’ advice defines code which is executed when the join point is reached and has control over whether the computation at the join point (i.e. an application method) is allowed to execute.

Aspect

An aspect is a modular unit of a cross-cutting implementation that is provided in terms of pointcuts and advices, specifying what (advice) and when (pointcut) its code is going to be executed. In terms of codification, aspects are similar to objects.

Aspect Weaver

The final application is generated by incorporating cross-cutting concerns into a final executable form and invoking a special tool called an aspect weaver. Figure 5-1 illustrates the concept of weaving in AspectJ where aspects and normal Java code are woven together and compiled into Java byte code.

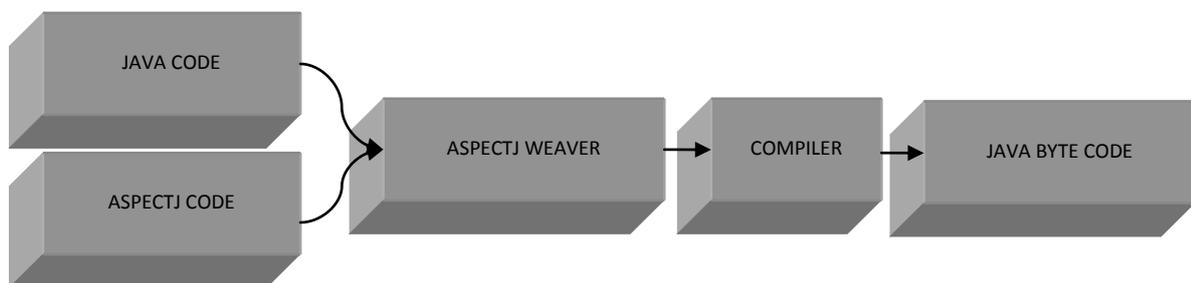


Figure 5-1: Illustration of the Weaving Concept

5.4 AOP Frameworks

The frameworks that are most widely known and used are AspectJ, AspectWerkz, JBoss AOP, and Spring AOP. The frameworks that are less popular are abc, aspect#, AspectC++, and JAC (Kersten, 2005; Wakaba, 2004)

AspectJ

AspectJ is an extension of the Java language syntax and semantics and provides its own set of keywords for developing aspects. In addition to containing fields and methods, AspectJ's aspect declaration contains pointcut and advice members

AspectWerkz

AspectWerkz, JBoss AOP and Spring AOP add aspect semantics without changing the Java language syntax. AspectWerkz provides two ways of making AOP declarations: Annotations and JavaDoc style declarations.

JBoss AOP

JBoss AOP has an XML-based aspect declaration style – aspect, pointcut and advice declarations are made in XML. Advice is implemented using plain Java methods that are invoked by the JBoss AOP framework.

Spring AOP

Spring AOP also uses an XML-based aspect declaration style, and similar to JBoss AOP, advices are implemented in a Java method with special parameters that are invoked by the Spring framework.

AspectJ was selected to develop the model concept presented in this thesis. It was assumed that as it is the most mature from all other aspect-oriented languages, it would be more thoroughly tested than other recent aspect-oriented compilers. Furthermore, since the AspectJ is more evolved than the other languages, it has a richer tool support. For example, the Eclipse platform which an integrated development environment, provides a mechanism to visualise an aspect-oriented system graphically.

5.5 Evaluating Aspect-Oriented Programming

Although aspect-oriented programming introduces an elegant implementation of separation of concerns, it does have its challenges. According to Murphy et al. (1999), the aspect-oriented approach makes it easier to reason about and develop ‘certain kinds of application code’; this implies that aspect-oriented programming cannot be applied in every situation. According to Padayachee and Eloff (2006), aspect-oriented programming is not only inappropriate in the programming of small-scale systems, but several cross-cutting concerns will have to be identified across a system to warrant the use of aspect-oriented technology. As this is a new technology, problems are to be expected in terms of testing and debugging the code. To start with, Alexander and Bieman (2002) maintain that as a result of the weaving process, isolating faults will be difficult as they may reside in the source code, aspect or woven code. Secondly, aspects being applied to pointcuts can interfere with performance and predictability of the program behaviour (Läufer et al., 2003).

Another challenge, as noted by Chen (2004) is that of understandability. A many-to-many relationship may exist between aspects and the primary abstractions they integrate with, which would potentially require an understanding of many other aspects to understand only one. According to Chen (2004), a complication may arise when several different authors each writes a collection of aspects to be woven. Each programmer must be au fait with the set of primary abstractions that his/her aspects can be woven with. This implies that each programmer must know about the other aspects that are used, either by direct composition or indirectly as a result of weaving. All of these activities will inevitably place a greater cognitive burden on aspect authors.

The advantages gained from using aspect-oriented programming include increased modularity, reduced development time, increased maintainability, improved reusability, more flexibility and simpler class hierarchies (Elrad et al., 2001). In addition to the benefits of reduced complexity and improved maintainability of software, programmers may be better able to understand an aspect-oriented program when the effect of aspect code has a well-defined scope (Walker et al., 1999). The presence of aspect code may also alter the

strategies that programmers use to address tasks perceived to be associated with aspect code (Walker et al., 1999).

As with all technology, several disadvantages have been identified despite all the great advantages. As aspect-oriented programming is a new technology, some issues are still unresolved. For instance, Alexander and Bieman (2002) pose several pertinent questions regarding aspect-oriented technology: 'How do we measure the complexity that results from the weaving process?'; 'Can we control or minimise the cognitive distance induced by the weaving process?'; 'How do we maintain aspect-oriented programs?' and 'How do we effectively test and analyse aspect-oriented programs?' Furthermore, Baniassad and Clarke (2004) and Clarke (2002) assert that identifying cross-cutting behaviour is difficult as it is entangled with other behaviours.

Murphy et al. (2001) on the other hand, pose questions providing several opportunities for research, such as: 'Does aspect-oriented programming work for large, multi-developer projects?'; 'To what kinds of problems is it best suited?' and 'What kinds of constructs are most usable for specifying crosscuts?'. Based on their empirical study, Murphy and his fellow researchers argue that aspect-oriented programming shows clear 'promise', but there is still much to learn about it. This implies that there is a definite need to carry out more evaluative studies on aspect-oriented programming. This need will be addressed in Chapter 8, where the aspect-oriented implementation of the model will be assessed against the object-oriented implementation.

Providing aspect-oriented solutions to an access control problem may provide greater insights into the paradigm and allow the access control problem to be viewed from a different perspective. Incorporating an aspect-oriented approach may also involve a shift in philosophy, as researchers and developers may have to discover innovative ways of maximising the positives of the technology by rethinking the access control in an alternative way. Just as the move from structured programming to object-oriented programming involved readdressing access control implementations, so too will the shift from object-oriented programming to aspect-oriented programming.

5.6 Conclusion

This chapter provided a brief overview of the evolution towards the aspect-oriented paradigm. The paradigm's goals, strengths and weaknesses were identified and briefly contrasted to those of the object-oriented paradigm. The object-oriented paradigm is, however, here to stay and it will not be replaced by aspect-oriented programming. In fact, aspect-oriented programming is merely a next step in the evolution of object-oriented technology and a refinement of object-oriented technology. Despite all the difficulties acknowledged and outlined in this chapter, aspect-oriented programming has been touted as a way to resolve security issues. Aspect-orientation is viewed as a better context in which to implement security concerns efficiently. The relationship between security and aspect-oriented programming will be explored in more detail in the next chapter.

CHAPTER 6:

ASPECT-ORIENTED SECURITY

6.1 Introduction

Security is a constant and pervasive concern in software systems. A major cause of this fact is the structural difference between application logic and security logic. A significant amount of work has been done in aspect-oriented security to warrant making the process more systematic in terms of software design and development.

According to Bodkin (2004) aspect-oriented security design 'is relevant for all the major pillars of security: authentication, access control, integrity, non-repudiation as well as the supporting administration and monitoring disciplines required for effective security'. For example, Alexander and Bieman (2002) state that 'code that implements a particular security policy would have to be distributed across a set of classes and methods that are responsible for enforcing the policy. However, with aspect-oriented technology, the code implementing the security policy could be factored out from all classes into an aspect.' Accordingly, the code that affects the implementation of multiple classes and methods is localised in one cohesive place, namely an aspect.

In addition to the dimension of abstraction offered by aspect-oriented programming, it also facilitates the implementation of additional security features so as to constitute a fully operational software system.

This ease of extensibility has the following advantages:

- It allows for better separation of concern and therefore better division of labour between application developers and security engineers.
- It also allows security to be added in a more agile manner since it is not necessary to consider it during requirements and specification phases.
- Using aspect-orientation to enforce security policy at compile time is also advantageous as it is a lot more efficient than code reviews (Boström, 2004).
- Crosscutting concerns can be added or removed without making invasive modifications to original programs (Ubayashi et al., 2004).

6.2 Aspect-oriented programming and its application to security

The discussion that follows highlights the relevance of aspect-oriented technology in terms of implementing some of the major pillars of security (access control and authentication, accountability and audit, data protection and information flow controls) in software systems.

6.2.1 Access Control and Authentication

Access control is a fundamental part of computer security where every requested access must be governed by an access policy stating who is allowed access to what. The request must then be mediated by an access policy enforcement agent (Pfleeger and Pfleeger, 2003). The seminal work in this area was conducted by De Win et al. (2002), who actually generalised the aspects they developed for access control to promote the reusability of these aspects. In an earlier publication (2001) they delineated three types of aspects, namely **Identification**, **Authentication** and **Authorisation** for access control in the aspect-oriented paradigm.

Program Listing 6-1: Generalised Aspect Code for Access Control

```
abstract aspect Identification of eachobject(entities()){
abstract pointcut entities();
public Subject subject null;
}
abstract aspect Authentication of eachcflowroot(authenticationCall()){
    private Subject subect;
    abstract pointcut serviceRequest();
    .....
}
abstract aspect Authorization{
    abstract pointcut checkedMethods();
    .....
}
```

The **Identification** aspect is used to tag the entities that must be authenticated, and as a container for identity information of the subject. The subject (see Program Listing 6-1) included in the aspect is used to determine whether access should be allowed or not. The **Authentication** aspect passes authentication information to the access control mechanism, while the **Authorisation** aspect checks the access based on the identity information received through the Authentication aspect.

Slowikowski and Zieliński (2003) considered how aspect-oriented security could enhance container-managed security and also demonstrated how identification, authentication and access control may be applied to components without modifying the source code. They concluded that aspect-oriented security required no modification of the application's source code to introduce security and that the procedure was highly flexible and extensible. The significance of their research is that if access policies are unknown or vague, access control features may be implemented after the development of other requirements or when these policies have been defined more clearly. Furthermore, the abstraction of access control policies eases security management and development significantly, as security experts may be allocated specifically to the development of these features.

According to Padayachee and Wakaba (2007), the aspect-orientated paradigm's versatility in terms of access control measures has been further validated by studies conducted within differing approaches to access control. The paradigm has been leveraged to implement discretionary access control (see De Win et al. (2002)), role-based access control (see Pavlich-Mariscal et al. (2005)) and mandatory access control (see Ramachandran et al. (2006) and Padayachee and Eloff (2007)).

6.2.2 Accountability and Audit

Accountability and audit serve to collect and analyse the activity of an information system. They aim at detecting security violations and defining causes, which may also be easily implemented with aspects (Slowikowski and Zielinski, 2003). In Program Listing 6-2 below, Slowikowski and Zieliński (2003) go on to demonstrate that an aspect could keep a log of exceptions thrown from a specific component – without modifying the component.

Program Listing 6-2: Demonstrating Accountability and Auditing with Aspect-Orientation

```
aspect BankAspect{
    pointcut bankMethods():
    execution (public *bank.*(..) && this (SessionBean);
    //Log information after throwing BankSecurityException from SessionBeans
    // which belong to the bank package
    after() throwing (BankSecurityException e): bankMethods(){
        Log log = Log.getInstance();
        log.write(e);
    }
}
```

6.2.3 Cryptographic Controls

In an experiment conducted by Boström (2004), it was found that by using aspect-oriented programming, database encryption could be added after the initial system had been completed. This case study showed that using aspect-oriented programming resulted in better modularity, database independence and less code. However, in certain instances the logic developers could not be totally alienated from the process of encryption, because the development of the functionality sometimes depended on the encryption process.

6.2.4 Information Flow Controls

There is some duality between access control and information flow control as both mechanisms are concerned with the flow of information. However, information flow control is more than access control, as an illegal flow may occur even when only authorised requests are performed on an object. As such, most access control models are supplemented with some form of information flow control.

Masuhara and Kawauchi (2003) found that although sanitising was a cross-cutting concern, there was no possible way to define a pointcut that would be able to detect whether a string was from an unauthorised source or not, or whether it contained unwanted information. Hence, they proposed a new pointcut called *dflow* that addresses the dataflow between join points as an extension to the AspectJ Language. Recall that join points are well-defined points where calls to aspect code would be inserted. The authors did not address security classifications and their dataflow definition only dealt with direct information flow. As no studies have been performed exclusively on this area, and seeing that aspect-oriented programming is an evolution in object-oriented programming, it would be pragmatic to investigate information flow control from this context as well (Padayachee and Wakaba, 2007).

6.2.5 Protection from invasive software

Aspect-oriented programming has been used to implement software tampering detection mechanisms in applications running on untrusted hosts (Falcarin et al., 2004). This involved the use of aspect-oriented programming to realise self-checking, a process where a program checks itself to verify that it has not been modified. In terms of evolving verification techniques as security threats change, using the separation-of-concerns principles makes it easier to 'swap in and out and evaluate alternative treatment options' (Houmb et al., 2004)

6.2.6 Security kernels

A security kernel is responsible for enforcing the security mechanisms of the entire operating system (Pfleeger and Pfleeger, 2003). Engel and Freisleben (2005) developed a tool for deploying dynamic aspects in the kernel space of an operating system. They determined that dynamic aspect-oriented programming was suitable in this area owing to the changing requirements, internal conditions and cross-cutting functionality of security kernels, and also found that the performance impact of using aspect-orientation was negligible in most instances.

6.2.7 Verification

Kumar et al. (2001) developed a framework that uses the aspect-oriented programming paradigm to verify that commercial off-the-shelf components are developed as per security contracts. Validating a component's performance, resource usage, transaction support, security, data persistency and distribution, are concerns that cross-cut a system's components – hence the aspect-oriented paradigm is appropriate for these purposes. Furthermore, as verification procedures tend to be similar in most applications, using the aspect-oriented paradigm facilitates the reusability of these validation measures (Grundy and Ding, 2002) via abstraction.

As discussed above, the aspect-oriented paradigm is revolutionising security implementations. It allows for security implementations to be adapted to changing needs more rapidly, and this is critical, given that security needs are impermanent. Furthermore, security solutions can seldom cater for all possible violations and these are usually discovered retrospectively. The added value of treating security as a separable concern is that it promotes reusability. This is significant, as the solution developed for the model concept presented in this research may require only slight modification for reuse in other information security systems in view of the fact that security aspects tend to be generic

6.3 Conclusion

In this chapter, the relevance of aspect-oriented programming to information security was explored and it became evident that it was in fact relevant to all pillars of security. Security concerns must inform every phase of software development, from the requirements phase through to the design, implementation and deployment (Devanbu and Stubblebine, 2000). During systems development, security requirements may be vague or they may change, and they may well be considered only as an afterthought after the system has been deployed. The extensibility of the aspect-oriented paradigm allows for these concerns to be implemented after other requirements have been developed. Furthermore, if a security concern has to be maintained due to the discovery of new security threats in the environment, the separation of concerns would fundamentally simplify this task. Within other programming paradigms, it may result in several fixes across several components, thus resulting in inconsistencies and regression faults.

The prototype that was designed to test the proof-of-concept of the model will be presented in Chapter 7 while Chapter 8 deals with its implementation, using the aspect-oriented approach. As the model is intended to augment traditional access controls, it is posited that the aspect-oriented approach will be highly suitable to fulfil this need without compromising the integrity of the system as a whole.



PART 3:

**CHAPTER 7:
THE OAC(UCON) MODEL**

**CHAPTER 8:
THE PROTOTYPING AND MODEL EVALUATION**

**CHAPTER 9:
CONCLUSION**

CHAPTER 7:

THE OAC(UCON) MODEL

7.1 Introduction

This chapter introduces the concept of the **Optimistic Access Control with Usage Control** model, designated the OAC(UCON) model. In the previous chapters the inadequacies of traditional access controls models were highlighted, and it became clear that the requirements for the OAC(UCON) had to include flexibility, adaptability and an open architecture. However, there should also be provisos that prevent abuse of the openness offered by this model. An example of the model that meets these requirements is presented at this juncture.

7.2 A motivating example

Suppose company ABC is an e-Recruitment company where clients and prospective candidates (job seekers) place job orders and applications respectively online. Company ABC then maintains databases of candidates and clients. While internally the company places access controls on sensitive information such as salaries, the information for collaborative job matching is unrestricted. Suppose an employee decides to download all the telephone numbers that are available on these databases and sells it to a telemarketer. Due to the lack of deterrents, this act is relatively easy to carry out. Furthermore, the employee may claim that he was unaware of the fact that his act was unethical. This type of security breach is typically blamed on the user and a lack of user training. The rationale behind such a security breach is contradictory to the security usability requirements that should be implemented in a software system. According to Zurko (2005) we have to ask, 'why did the system make the insecure option so easy and attractive' – and in this case ultimately lucrative? Perhaps if

synchronous system deterrents had been deployed at the time of usage, the employee may have been deterred from carrying out an illegitimate act.

In this scenario, the following stipulations or mechanisms could have been used as usage control deterrents:

- **Pre-obligation:** The user must click on a button in a window to indicate that he/she agrees not distribute this information.
- **Ongoing obligation:** A window with the following warning '*This dataset must be used exclusively for work-related purposes ONLY*' is to remain open at all times.
- **Pre-condition:** The user is warned beforehand that 'this information may be accessed during business hours only'.
- **Ongoing conditions:** The information may be accessed during business hours only (same as the pre-condition as it is time dependent). However, although the user may have accessed the information during business hours (i.e. initially satisfying all the conditions), the ongoing condition may become invalid as time passes.
- **Post-obligation:** The user must indicate the priority of the task completed if he/she actually accessed these databases outside business hours. This information could be used to develop a profile on the user.

The post-obligation implies that an employee may in fact access the databases after hours. Under the optimistic paradigm the employee should ultimately be able to download the data in the case of an emergency. This is permitted, as the employee should not be hampered in the performance of his/her duties. While fulfilling the post-obligations facet is within a user's control, the pre-conditions and ongoing conditions are not. For this reason a break-the-glass tool may be included to allow for overriding the pre-conditions or ongoing conditions.

The Break-the-Glass policy

The Break-The-Glass (BTG) policy provides a mechanism to override access control policies (Ferreira et al., 2006) as part of the access control policy stated in the previous section. This act can be justified because there are situations when access is required, even if it means

that confidentiality is breached. The important issue is that this breach is openly declared to the responsible parties and the access is properly analysed afterwards. At that stage it can then be considered whether the breach was well justified or whether it was an intrusion.

The following section investigates how such deterrents may be practically implemented under the optimistic paradigm.

7.3 Architecture

Many systems are based on a three-tiered architecture – access is via the web, the application programs reside within an application server, and the data is stored within a database system (Li et al., 2005). Only the application tier is considered in the next section (Figure 7-1).

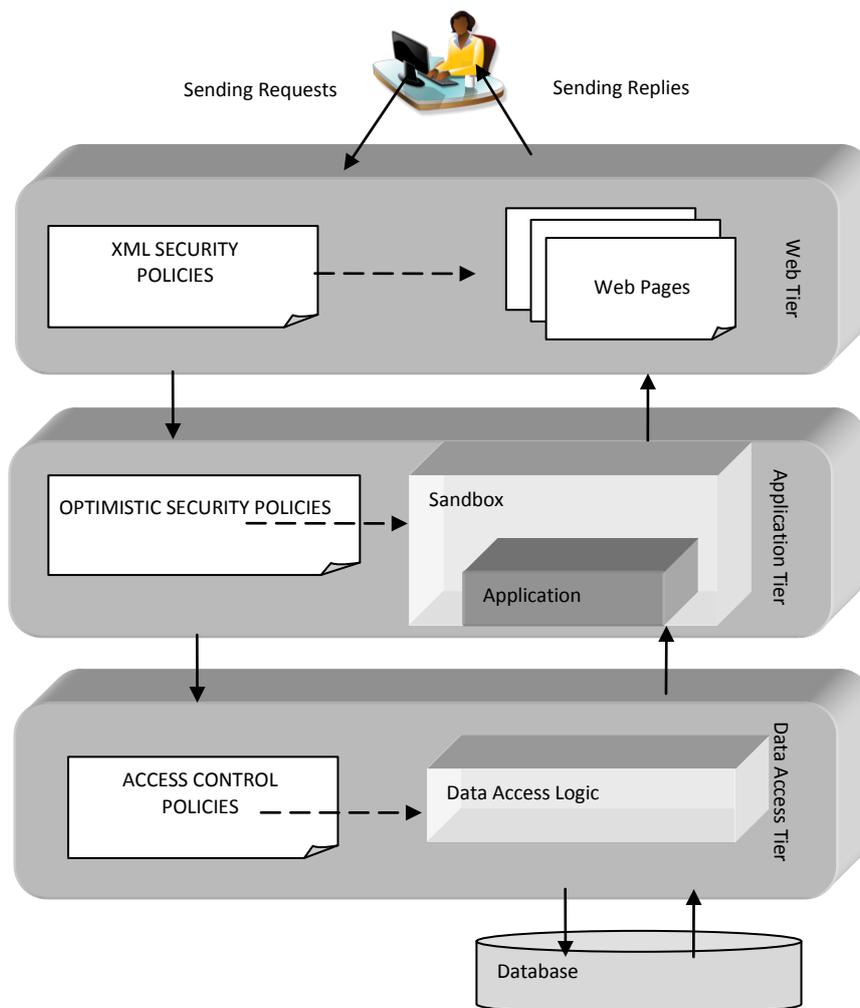


Figure 7-1: Architectural Diagram

In Chapter 8 the prototype is designed within the Application Layer where it is protected by the Sandbox offered by the Java Authentication and Authorisation Service (JAAS). In the basic Java security model, trusted code is allowed full access to the system, and untrusted code is forced to execute in a restricted environment called the sandbox. The access control policies of the sandbox are established by an instance of the `SecurityManager` class. A security manager provides methods that determine whether a particular operation is permissible (Tymann and Schneider, 2008).

The conceptual structure of the OAC(UCON) model is shown in Figure 7-2. The **Usage Enforcement Facility** includes the Customisation Module, which can be used to constrain a user's access to specific components of an object. Within the model, it is presumed that as the user's optimistic rights are downgraded, the data that he/she is allowed to access may be constrained according to sensitivity levels. The Monitoring Module involves logging all accesses, while the Update Module is involved in changing the access rights.

The **Usage Decision Facility** includes the Condition, Obligation, Break-the-Glass and Authorisation Modules. In general, most accesses are allowed – except for those users who breached the system in the past. The Authorisation Module therefore determines the level of access that is permitted. For example, an authenticated user is permitted to view data of all sensitivity levels. In most cases users are trusted to view all objects that are given optimistic access rights. The Condition Module decides whether conditional requirements for authorised requests are satisfied or not by using usage rules and contextual information (such as current time, IP address, etc.). The Obligation Module decides whether certain obligations have to be performed or not, either, before, during or after the requested usage has been performed. If there exists any post-obligation that has to be performed, this must be monitored by the Monitoring Module and the result has to be resolved by the Update Module in the Usage Decision Facility. If the user does not satisfy the obligations before or during the access, he/she is not permitted to access the information. However, when the conditions are invalid, the user is allowed to access the information by using the Break-the-Glass Module. This Module is intended for emergencies only and the user is clearly informed

about the consequences of accessing this information illegitimately. Access by ‘breaking the glass’ is always red flagged for auditing purposes.

The **Usage Protection Facility** is used to protect the integrity of the information. The Audit Module will check through all accesses and identify possible illegal accesses. It will additionally create a list of red-flagged accesses. If the user cannot justify an illegal access, then the Authorisation Module will restrict the user's optimistic access rights in future. This may also involve punitive action. If the user has performed some illegal modification to the data, the Roll-back Module will attempt to return the data back to its original state.

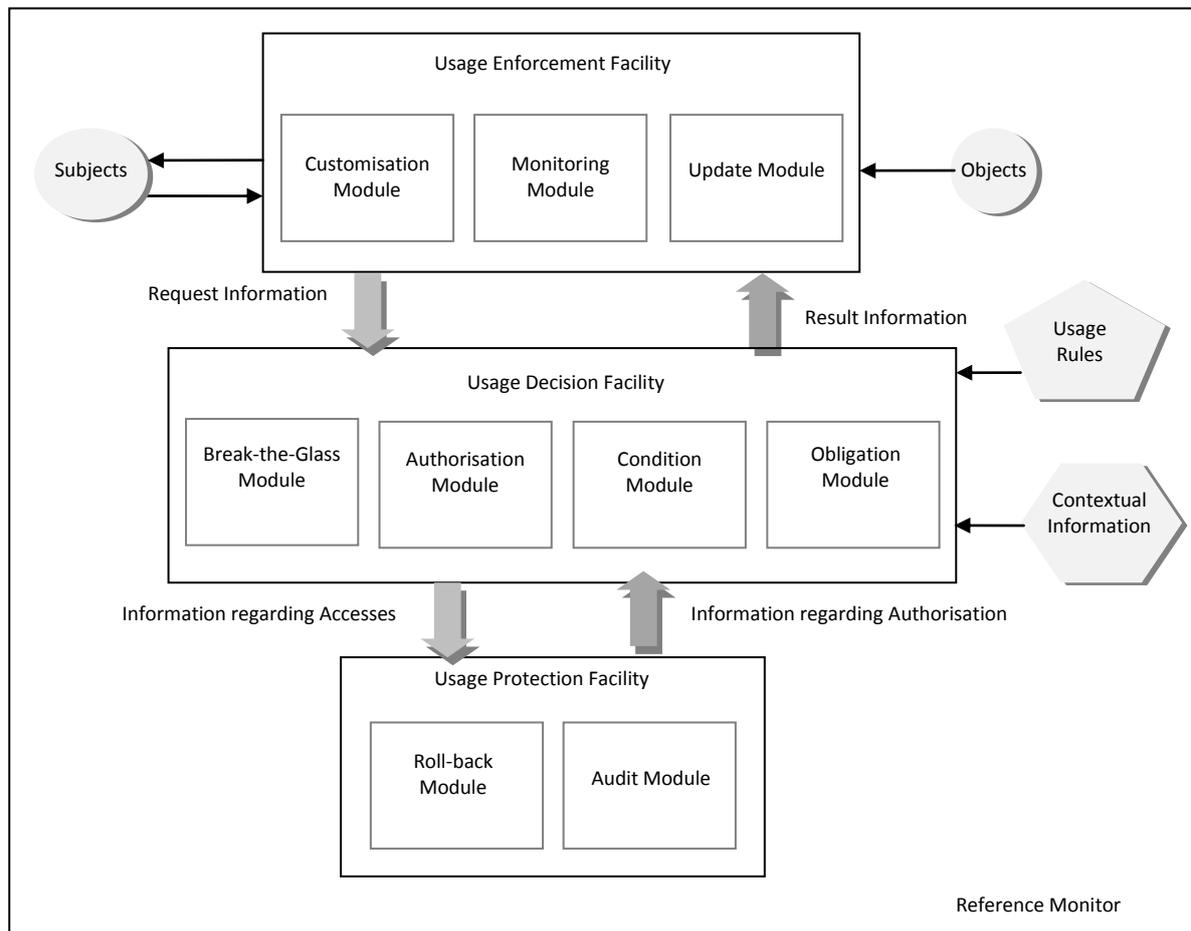


Figure 7-2: Conceptual Structure for Optimistic Access Control enhanced with Usage Control

The Break-the-Glass Module, which provides a mechanism to override access control policies, is used in the same way as by Ferrieria et al. (2006). It is important to note that the responsible parties are fully aware of this security breach and that the access is properly analysed afterwards so as to consider whether the breach was well justified or a mere intrusion. What is different from the way in which the Break-the-Glass policy is used by Ferrieria et al. (2006) is that with the OAC(UCON) Model it is enforced when the system conditions are not satisfied. For instance: users are only allowed to access data from 8:00am to 5:00pm. In an emergency, the user will be allowed to override this system condition by making use of the Break-the-Glass facility.

The sequence of a user's interaction with the OAC(UCON) model can be summarised as follows:

1. The Authorisation Module checks the constraints on the information requested by using the Customisation Module. In general, the user is allowed to access the information if he/she is authenticated and the data is under the optimistic access control domain.
2. Before the user is allowed access, he/she has to fulfil one or more specific pre-obligations set up by the Obligation Module. If these obligations are not met, the user is not allowed to access the information.
3. If the system pre-conditions are valid, the user is allowed to access the information. These permissions are maintained by the Condition Module. However, if the system conditions are not valid, the user is presented with an opportunity to use the Break-the-Glass facility deployed by the Break-the-Glass Module. The user is given adequate warning about the usage of this facility. Moreover, it is red-flagged and logged.
4. The system is frozen until the user decides whether or not to use the Break-the-Glass facility.
5. If the user employs the Break-the-Glass facility, the system verifies whom it needs to notify and proceeds with the access.
6. All notifications and user actions are registered automatically by the Monitoring Module.

7. During the access, the user has to meet the ongoing obligations. If they are not met, the user's access to the information is automatically revoked.
8. During the access, the ongoing conditions may become invalid. The user then has the opportunity to use the Break-the-Glass facility to continue with the access.
9. After the access, the user may have to satisfy a post-obligation. This will be monitored by the Monitoring Module.
10. The Update Model may change the access rights of a user if he/she has committed an unjustified breach.
11. The Audit Model checks for red flags and unjustified breaches.
12. The Roll-back Module may be deployed if an unjustified breach resulted in the data being compromised.

The OAC(UCON) model is intended to fit into a mixed-initiative access control framework (see Figure 7-3), encompassing traditional access control such as role-based access control for highly classified information and optimistic access control enforced with usage control for information that is unclassified. Under the optimistic access control paradigm, users should be allowed access under most circumstances. Thus, access would not depend on the subject attributes or the object attributes. It is assumed that data is freely available and not subject to authorisation unless the user's optimistic rights have been downgraded. While the user will not be permitted to access the information unless the obligations are satisfied, he/she will under special circumstances be allowed to access the data by utilising the Break-the-Glass facility – even if the pre-condition(s) or ongoing condition(s) are invalid.

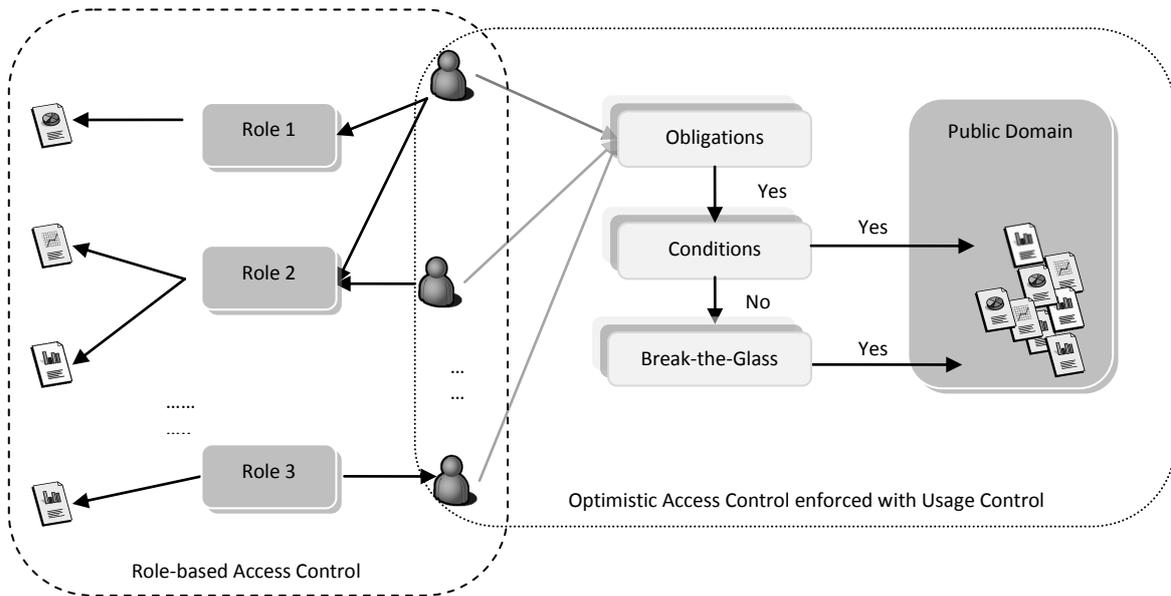


Figure 7-3: A Mixed-Initiative Access Control Framework – combining RBAC with OAC(UCON)

7.4 Detailed Design

7.4.1 Formal Specifications

The formalised definition of the optimistic access control enhanced with usage control is as follows:

Pre-Authorisation:

- Subjects S , Objects O , Rights R
- Function that checks if the user s , requesting access o with right r has Optimistic Rights:
 $OptimisticRights(s,o,r)$
- Function that checks if the set of Pre-Conditions are satisfied or not:
 $PreC(s,o,r)$
- Function that checks if the set of Pre-Obligations are satisfied or not:
 $PreB(s,o,r)$

- Function that requests if the user wants to override pre-conditions or not by using the Break-the-Glass facility:

Break-the-Glass(s,o,r)

- Access is allowed momentarily if the pre-obligations and pre-conditions are satisfied:

$$\begin{aligned} \text{allowed}(s,o,r) \Rightarrow & \text{OptimisticRights}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \\ & \wedge \text{PreB}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \\ & \wedge \text{PreC}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \end{aligned}$$

- Otherwise allow the user to use the Break-the-Glass facility

$$\begin{aligned} \text{allowed}(s,o,r) \Rightarrow & \text{OptimisticRights}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \\ & \wedge \text{PreB}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \\ & \wedge \neg \text{PreC}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \\ & \wedge \text{Break-the-Glass}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \end{aligned}$$

- Access is revoked if the pre-obligations are not met or if the Break-the-Glass is not used when the pre-conditions are invalid:

$$\begin{aligned} \text{revokeAccess}(s,o,r) \Rightarrow & \neg \text{Break-the-Glass}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \\ & \vee \neg \text{PreB}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \end{aligned}$$

Ongoing Authorisations

- Subjects S , Objects O , Rights R
- Function that checks if the set of ongoing Conditions are satisfied or not:

onC(s,o,r)

- Function that checks if the set of ongoing Obligations are satisfied or not:

onB(s,o,r)

- Function that requests the user to override the ongoing conditions or not, using the Break-the-Glass facility:

Break-the-Glass(s,o,r)

- Sustain access if the ongoing obligations and ongoing conditions are momentarily satisfied:

$$\begin{aligned} \text{allowed}(s,o,r) \Rightarrow & \text{onB}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \\ & \wedge \text{onC}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \end{aligned}$$

- Otherwise allow the user to employ the Break-the-Glass facility if the ongoing conditions are invalid in order to sustain access:

$$\begin{aligned}
 \text{allowed}(s,o,r) \Rightarrow & \quad \text{onB}(s,o,r) & \rightarrow & \quad \{\text{true}, \text{false}\} \\
 & \quad \wedge & & \\
 & \quad \text{-onC}(s,o,r) & \rightarrow & \quad \{\text{true}, \text{false}\} \\
 & \quad \wedge & & \\
 & \quad \text{Break-the-Glass}(s,o,r) & \rightarrow & \quad \{\text{true}, \text{false}\}
 \end{aligned}$$

- Access is revoked if the ongoing obligations are not met or if the Break-the-Glass facility is not accepted when the ongoing conditions are invalid:

$$\begin{aligned}
 \text{-revokeAccess}(s,o,r) \Rightarrow & \quad \text{-onB}(s,o,r) & \rightarrow & \quad \{\text{true}, \text{false}\} \\
 & \quad \vee & & \\
 & \quad \text{-Break-the-Glass}(s,o,r) & \rightarrow & \quad \{\text{true}, \text{false}\}
 \end{aligned}$$

Post Authorisation

If a user does not satisfy their post-obligations or is involved in an unjustified breach, then the user's optimistic rights are downgraded and his/her access will be constrained in future.

PostUpdate (OptimisticRights (s,o,r)): OptimisticRights (s,o,r) \Rightarrow

$$\begin{aligned}
 & \quad \text{-PostObligations}(s,o,r) & \rightarrow & \quad \{\text{true}, \text{false}\} \\
 & \quad \vee & & \\
 & \quad \text{-JustifiedBreach}(s,o,r) & \rightarrow & \quad \{\text{true}, \text{false}\}
 \end{aligned}$$

Administrator-controlled attributes can be modified by administrative actions. These attributes are modified at the administrator's discretion but are 'immutable' in that the system does not modify them automatically. Mutable attributes are automatically modified by the system (Park et al., 2004). For instance, if the users cannot justify having used the Break-the-Glass procedure, their optimistic access rights may be revoked or constrained in future.

7.4.2 The Use Case Diagram of Usage Control under the Optimistic Access Control Paradigm

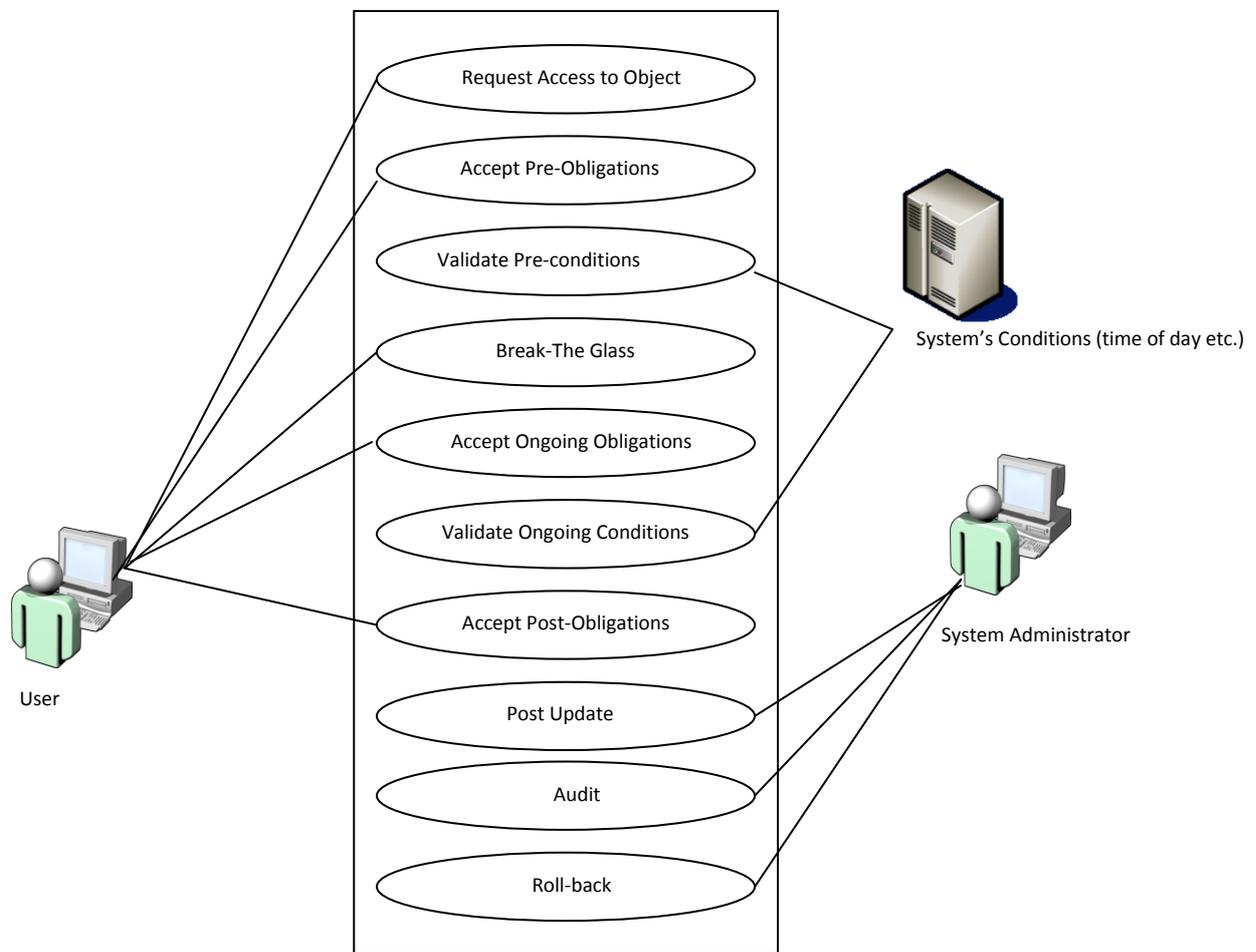


Figure 7-4: Use Case Diagram of OAC(UCON)

The implementation of pre-authorisation is relatively simple as it warrants checking the conditions and obligations before the user may proceed. The implementation of ongoing authorisation is, however, non-trivial. The implementation decision taken to deal with this issue is to be addressed in Chapter 8. As the OAC(UCON) model is based on optimistic access control, most users are trusted to access information that is relevant to their context. Note that this is a less prescriptive approach than, say, role-based or mandatory access control. Due to the openness of the architecture, the user has to meet all pre-obligations and ongoing obligations, as he/she is expected to behave in a trustworthy manner.

In terms of the enforcement of security policies, it is imperative that this function be located centrally and enforced uniformly. The same notion would apply to the implementation of such policies in terms of application logic (Verhanneman et al., 2005). This type of deployment may be achieved through the use of aspect-oriented methodologies. The premise of the model is to create an aspect that will intercept calls when a subject requests access to an object and to enforce optimistic access control enhanced with usage control. A significant amount of work has been conducted in aspect-oriented security in respect of access control. The implementation of access control using aspect-oriented programming has been shown to ease the development of security-type concerns such as access control (De Win et al., 2001), as it results in an implementation that is easier to maintain and port to different environments.

According to Jones and Rastogi (2004), security controls may fall in one of four categories: *corrective control*, *deterrent control*, *detective control* and *preventative control*. Access control falls in the preventative control category. Information under this protection is typically secured in terms of roles or attributes. However, information under the public domain is not. Securing a distributed computing environment against malicious or otherwise disruptive use involves two aspects (Georgiev and Georgiev, 2001):

- Social, where the safeguarding of a computer system relies on social deterrents, such as shameful exposure or prosecution.
- Technical, where the system is protected by technical means, such as encryption algorithms and access controls.

Detective functions attempt to identify unwanted events while they are occurring or after they have occurred. Recovery controls restore lost computing resources or capabilities and help the organisation to recover monetary losses caused by a security violation. Corrective controls either remedy the circumstances that allowed the unauthorised activity or return conditions to what they were before the violation (Kim and Leem, 2004). Deterrent controls are intended to discourage individuals from intentionally violating information security policies or procedures. Typically, organisations implement deterrents such as anti-virus systems, passwords or they foster security awareness. However, with the use of the OAC(UCON) model, deterrents are achieved in a proactive manner.

7.5 Conclusion

This chapter presented the OAC(UCON) model together with specifications for a practical and implementable system. One of the criticisms levelled at the model in earlier presentations was its applicability. This problem was addressed by viewing the model within a mixed-initiative context, that is, within the context of traditional access controls. The model is highly applicable in contexts where some data cannot be reasonably protected by traditional access controls and needs to be openly available. This data can now be relegated to the public domain but remain subject to usage control rules under the optimistic access control paradigm. In the next chapter, the prototype will demonstrate a subset of the functionality of the OAC(UCON) that focuses more on the usage decisions component of the model. The prototype is implemented by using aspect orientation to demonstrate the suitability of the paradigm for this context. In addition, the model concept is evaluated in terms of the design science research methodology to assess its effectiveness and scalability.

CHAPTER 8:

PROTOTYPING AND MODEL EVALUATION

8.1 Introduction

The literature survey presented in Chapters 2, 3 and 4 has led to the development of a Usage Control Model for Optimistic Access Control (OAC(UCON) model) (Chapter 7). The current chapter presents the design of a proof-of-concept prototype to demonstrate a subset of the model concept, based specifically on the usage decision. The prototype was developed using an aspect-oriented programming language namely AspectJ. In addition, the model was evaluated in terms of the design science research method (March and Smith, 1995) to test its scalability and efficacy as a security measure. During this process, several evaluative prototypes were developed so as to verify the model concept for commercial systems. Hence, the evaluative prototype encompassed a larger subset of the model concept than the proof-of-concept prototype. The programming paradigm for the evaluative prototypes was not stipulated. Consequently, the evaluative prototypes were implemented by using various design techniques and programming languages in order to provide a more comprehensive assessment of the model.

8.2 The aim of the proof-of-concept prototype

The aim of the prototype was to provide key working points to show how the model can scale up using aspect orientation. The prototype was developed using AspectJ to evaluate the aspect-oriented paradigm. Furthermore, for comparative purposes, the proof-of-concept prototype was developed using Java as an object-oriented language and it focused specifically on the **Usage Decision Facility** of the OAC(UCON) model. Section 8.3 is a formalisation and consolidation of an earlier publication by Padayachee and Eloff(2009).

8.3 Implementation of the proof-of-concept prototype

Figure 8-1 shows the activity diagram for the OAC(UCON) model. In terms of the formal specification presented in Section 7.4.1, this relates to the pre-authorization and ongoing authorisations as well as to the post-authorization formulations.

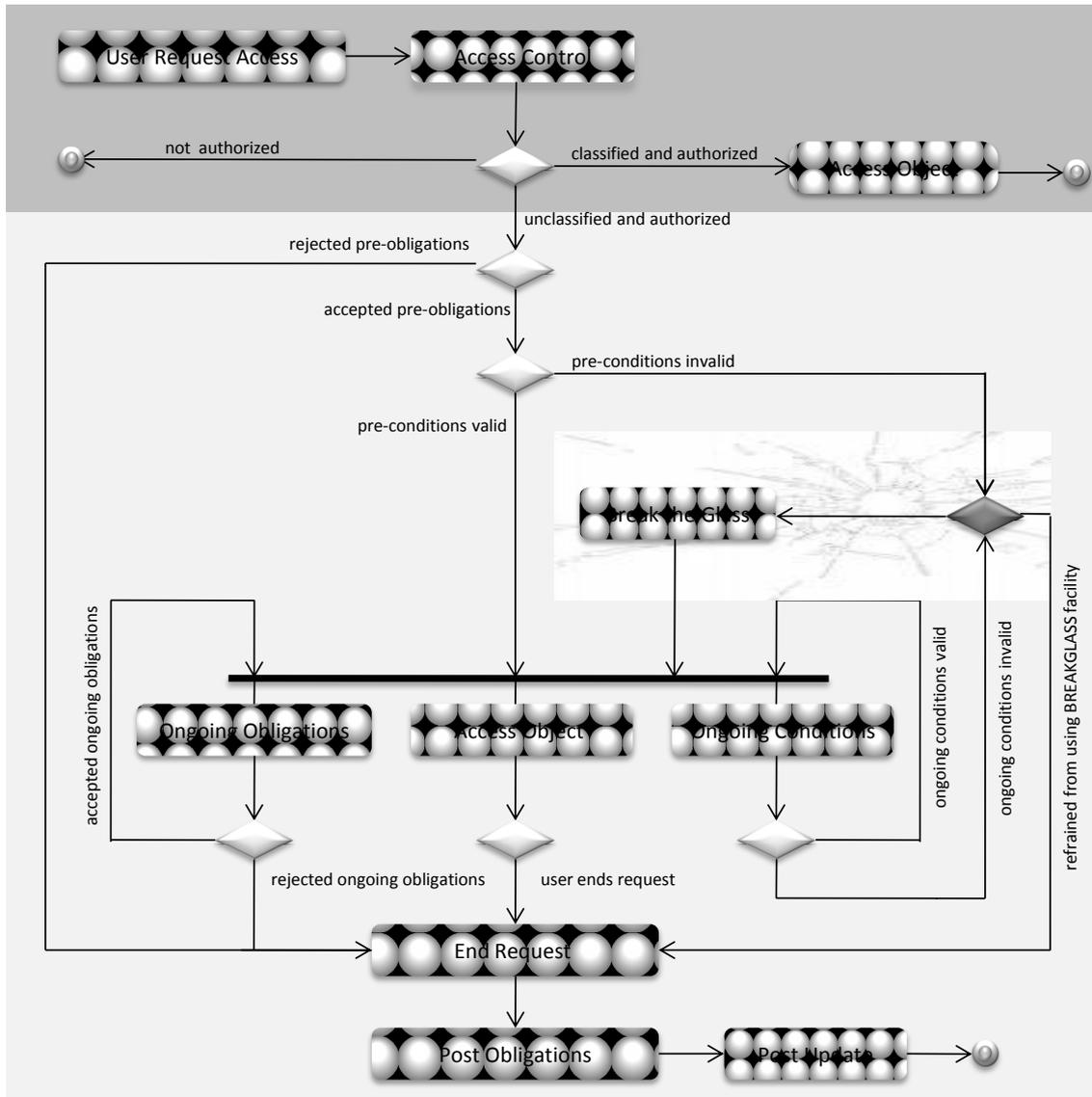


Figure 8-1: State Activity diagram of OAC(UCON) Model)

When a user requests access to an object, authorisation is performed by utilising subject and object information (attributes). Usage rules are used to check whether the request is permissible and whether the data is classified and subject to access control. If the data is in the public domain and hence unprotected by access controls, then the usage deterrence

mechanisms are deployed. Otherwise, the access control proceeds as expected with traditional access control based on user attributes.

Multithreading was used to implement the ongoing authorisations (see Figure 8-2). If a subject requests an object (such as a file), the pre-conditions and pre-obligations are checked and then two separate threads are invoked, representing the ongoing conditions and ongoing obligations respectively.

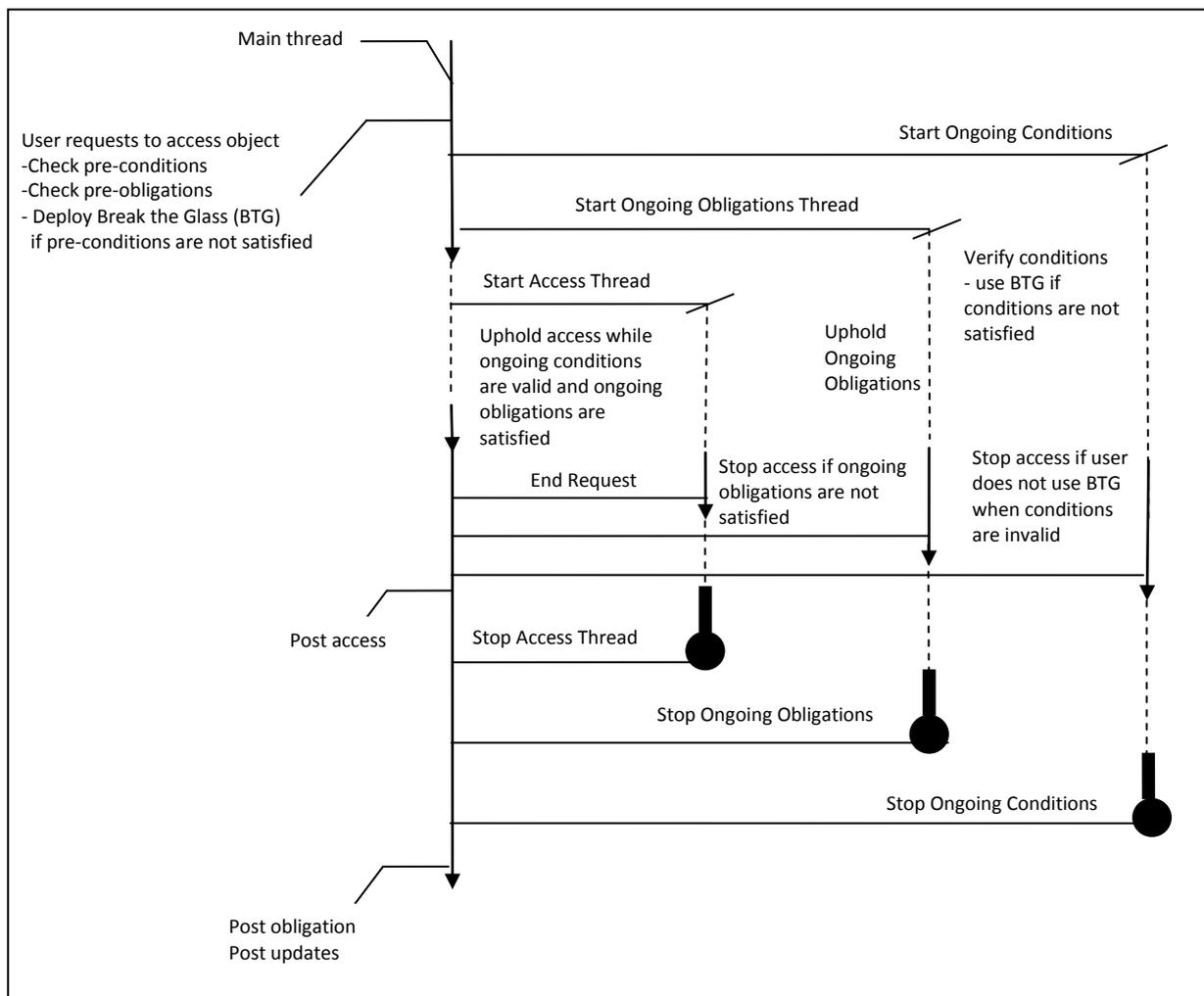


Figure 8-2: Thread Diagram of the OAC(UCON) model

As this model is based on optimistic access control, most users are trusted to access information that is relevant to their context. However, owing to the openness of the architecture, the user has to meet all pre-obligations and ongoing obligations. As the user is expected to behave in a trustworthy manner, the trust is maintained by the user's acceptance of the pre-obligations and ongoing obligations that are coupled with accessing the information. However, if the pre-conditions or ongoing conditions are invalid, the user is allowed to deploy the Break-the-Glass facility to access the information.

If the user does not accept the pre-obligations, he/she is not allowed to access the information at all. If the user does not accept the ongoing obligations, access is revoked. These refusals are not considered as breaches of trust. On the other hand, if the user accessed the information and then refuses to accept the post-obligations, such refusal is considered to be a breach of trust.

In the previous chapter, aspect-orientation was proposed as means of implementing usage control without making invasive changes to the code base of a fully operational system. To illustrate the utility of the aspect-orientation in terms of the enforcement of usage control, a scenario is considered where a fully operational software system that performs traditional access control has to be complemented with the features offered by the OAC(UCON) model during post-delivery maintenance. In this scenario, the class `SampleAuthorization`, as shown in Program Listing 8-1, controls user accesses to objects by using traditional access controls. An object could be a file 'c:\candidate.txt'.

Program Listing 8-1: 'SampleAuthorization' class

```
public class SampleAuthorization implements PrivilegedAction {  
  
    public Object run() {  
        .....  
        AccessObject Access (SubjectName, ObjectName, "READ");  
        AccessObject.request ();  
        .....  
    }  
}
```

As shown in Program Listing 8-1, the execution point where the `request()` method is called involves checking whether this access request by a subject to an object is permissible. The operational system has classes that represent the access and access information called `Access` and `AccessInformation` respectively, where the `Access` class extends the `AccessInformation` class. The `AccessInformation` class represents the nature of the access, in terms of the subject-object attributes and the type of access. The Unified Modelling Language (UML) diagram provided in Figure 8-3 shows how these classes may be integrated with usage control features without modifying the operational system.

Since Aspect-Oriented Programming is a relatively new paradigm, there are no specific standards available in terms of designing aspects with regard to the UML. A position paper by Groher and Schulze (2003) was used to produce the UML diagram (Figure 8-3). The symbol  was created to show an intertype declaration.

As is shown, the Ongoing Obligations, Ongoing Conditions and Break-the-Glass mechanisms are represented by classes `OngoingObligation`, `OngoingCondition`, `BreakTheGlass`; and would inherit from the `AccessInformation` class as these classes require information about the access. The `UsageControlInjector` collaborates with these classes to perform its usage control functionality when a subject requests access to an object. As the `Access` class was not intended to cater for instances where access needs to be revoked owing to invalid usage control obligations and conditions, an `intertypeTypeInjectorOnAccess` aspect was included to perform the requisite actions when access needs to be terminated. For the sake of readability, only core classes that are affected by the aspects are shown. The `Sampleauthorization` class is not shown in the diagram as the `UsageControlInjector` provides its supplemental functionality around the `request()` method found in the `Access` class which is actually responsible for mediating a request between a subject and an object.

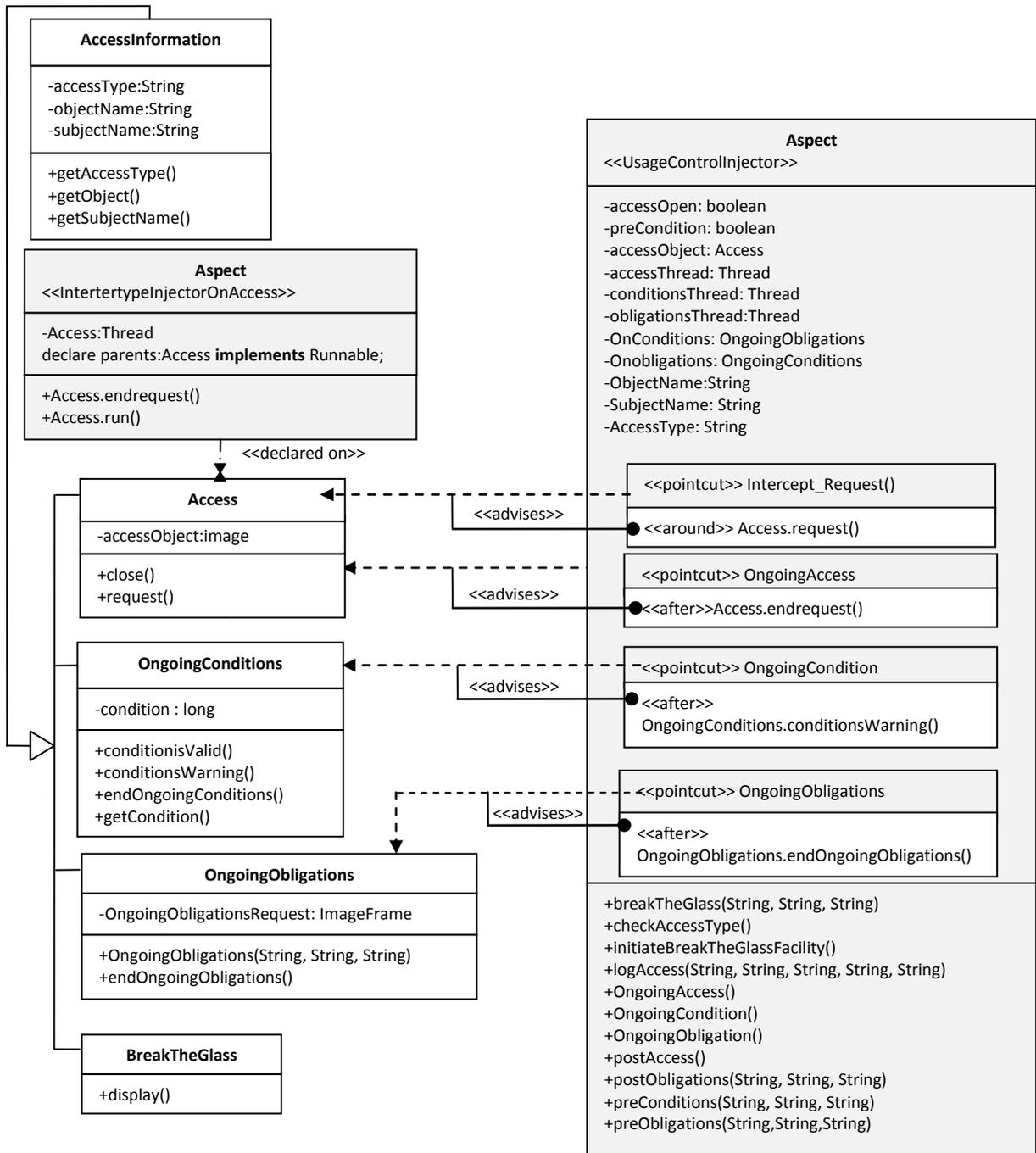


Figure 8-3: UML Diagram showing Aspect UsageControlInjector and Core Classes

8.4 An implementation overview of the proof-of-concept prototype

With aspect-oriented programming, the existing authorisation module which has methods for traditional access control can be augmented with usage control without modifying the source code. Furthermore, all details relating to usage control can be confined in a singular modular structure, namely an aspect, without it being mixed in with this module. To accomplish this, a generic aspect `UsageControlInjector` that delineates four pointcuts was defined. The first pointcut `Intercept_Request` intercepts those calls where a user requests access to an object, i.e. during programs execution where the method body of `request()` executes. The naming of the request method may differ from application to application. The `!within` expression is used to prevent infinite recursive calls. If no special precautions are taken, aspects that advise other aspects can easily and unintentionally advise each other recursively. The `target` keyword has to ensure that the executing code belongs specifically to class `Access`. The *around* advice defines code that is executed around the `request()` method when it is called. This advice initially determines if this information is in the public domain (i.e. controlled by Optimistic Access Control), otherwise it allows traditional access control to proceed as usual. The advice also contains operations to test the `preObligations` and `preConditions`. If the `preObligations` are not satisfied, the user is not allowed to access the object. If the `preConditions` are invalid, the user has the opportunity to use the Break-the-Glass facility to access the feature. The aspect invokes threads to maintain the ongoing conditions and ongoing obligations to control the request to use the object.

Program Listing 8-2: Showing the UsageControlInjector Aspect

```
public aspect UsageControlInjector {
    private static boolean accessOpen;
    private static boolean preCondition = true;
    private Thread obligationsThread;
    private Thread conditionsThread;
    private Thread accessThread;
    private Access accessObject;
    private OngoingConditions OnConditions;
    private OngoingObligations Onobligations;
    private String SubjectName;
    private String ObjectName;
    private String AccessType;

    pointcut Intercept_Request(Access AccessObject) :
execution(* *.request(..) && !within(UsageControlInjector)
&& target(AccessObject) ;
    void around (Access AccessObject )
    : Intercept_Request(AccessObject){
        accessOpen = true;
        accessObject = AccessObject;
        SubjectName = accessObject.getSubjectName();
        ObjectName = accessObject.getObject();
        AccessType = accessObject.getAccessType();
        if (OptimisticRights()){
            if (preObligations(SubjectName, ObjectName, AccessType)){
                if (preConditions(SubjectName, ObjectName, AccessType)
                    || breakTheGlass(SubjectName, ObjectName, AccessType)){
                    //start access Thread
                    //start OngoingObligation Thread
                    //start OngoingCondition Thread
                    while(accessOpen){
                        //wait
                    }

                    postObligations(SubjectName,ObjectName,AccessType);
                }
            }
        }
    }

    public boolean checkAccessType(){
        //determine whether this information is subject to optimistic access
        control
        return true;
    }

    //when there is conditions warning
pointcut OngoingCondition(): call(* *.conditionsWarning(..) ) &&
target(OngoingConditions);
after(): OngoingCondition(){
    initiateBreakTheGlassFacility();
}

    // when the user ends ongoingObligations
pointcut OngoingObligation() : call(* *.endOngoingObligations(..) ) &&
target(OngoingObligations) && !within(UsageControlInjector);
after(): OngoingObligation(){
    postAccess();
}
}
```

```
// when the access ends
pointcut OngoingAccess() : call(* *.endrequest(..) ) && target(Access) &&
!within(UsageControlInjector);
after(): OngoingAccess(){
    postAccess();
}

void initiateBreakTheGlassFacility(){
//method to initiate the Break-the-Glass facility
}

boolean preObligations(String SubjectName, String ObjectName, String
AccessType){
//method to perform preObligations
}

boolean preConditions(String SubjectName, String ObjectName, String
AccessType){
//method to perform preconditions
}

void postAccess(){
//clear up
//update logs
}

public void postObligations(String SubjectName, String ObjectName, String
AccessType){
//method to perform postObligations
}

boolean BreakTheGlass(String SubjectName, String ObjectName, String
AccessType){
//perform Break-the-Glass facility
}

void logAccess(String SubjectName, String ObjectName, String AccesType,
String Notice, String RedFlag){
//write to log file
}
}
```

The next two pointcuts, `pointcut OngoingCondition` and `pointcut OngoingObligation`, intercept execution points that indicate that the ongoing conditions and ongoing obligations are no longer satisfied. The *after* advices of each pointcut define code that is executed after such an irregularity is detected. In this case, if some action results in the `ConditionsWarning` or an `endObligations` method being called on either the `conditions` object or the `obligations` object, then this call will be intercepted by these pointcuts. If the ongoing obligations are no longer satisfied, then the access is revoked immediately. If the ongoing conditions are no longer satisfied, the user has the opportunity to use the Break-the-Glass facility to continue with the access.

The last pointcut is used to intercept execution points where the access is terminated by the user. Subsequently post-access activities such as logging the access are performed and the Ongoing Conditions and Obligations threads are ceased. (Appendix F provides details on the AspectJ semantics.)

The `Access` class would require additional functionality to mediate users' requests to an object and to end requests when required. This functionality may not have been provided in a way that fits in with the current usage of the `Access` class. To maintain the integrity of the class, aspect orientation permits a seamless integration of this additional functionality by facilitating the creation of a special aspect known as an *intertype declaration* without modifying the `Access` class. The *intertype declaration* construct is supported by aspect-oriented programming languages such as AspectJ. An *intertype declaration* is generally used to add on information such as methods or fields to an object without modifying the existing class. Furthermore, as this process needs to be controlled within thread, in Java it implies that this class must implement the `java.lang.Runnable` thread interface. With AspectJ, this can be done using the `declare parents` syntax so that `Access` class can be an active object. The relevant classes are provided in Appendix C.

Program Listing 8-3: Depicting an InterTypeDeclaration Aspect

```
public aspect IntertypeDeclarationOnAccess {
    declare parents: Access implements Runnable;
    Thread Access.aThread;
    public void Access.endrequest() {
        aThread = null;
        close();
    }
    public void Access.run() {
        //perform the request
    }
}
```

The `UsagecontrolInjector` is relatively generic as it can be reused within other contexts as well. Only the method specified that performs the access control, namely `request()`, would have to be re-specified according to the system naming. The `IntertypeDeclarationonAccess` aspect is partially generic, as the class name

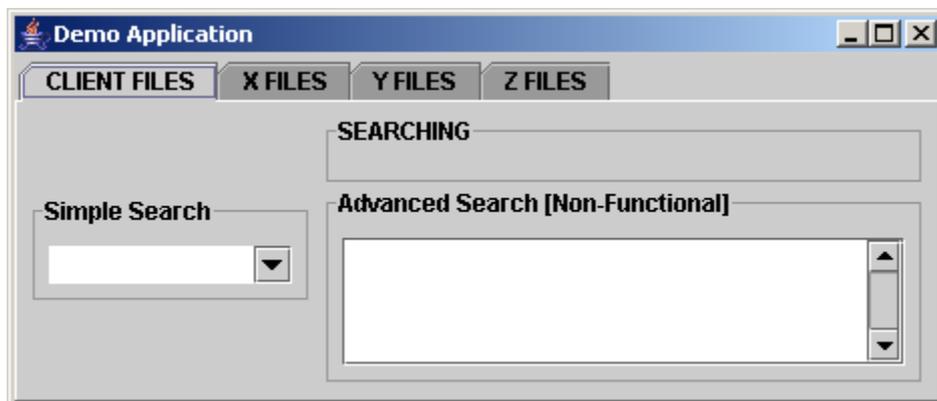
Access would have to be re-specified accordingly. Unlike typical instrumentation, using the aspect-oriented paradigm results in better consistency, as all methods that match the pointcuts are identified. Evidently, it is possible to augment usage control features in a system without modifying the existing system.

8.5 Proof-of-concept prototype operation

Step 1: Login and authentication of the user



Step 2: An application interface appears which allows searching of files.



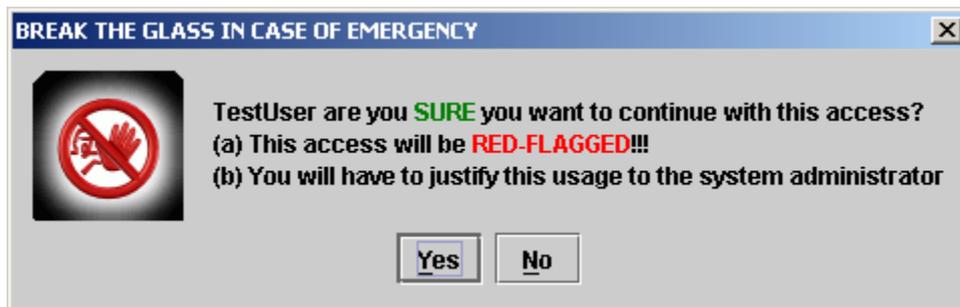
Step 3: User selects a file to open and the pre-obligation dialogue box opens. User has to accept the pre-obligation to move on to the next step.



Step 4: Pre-conditions warning appears if the pre-conditions are NOT met.



Step 5: The Break-the-Glass facility is invoked. User has to accept the Break-the-Glass option in order to access the file.



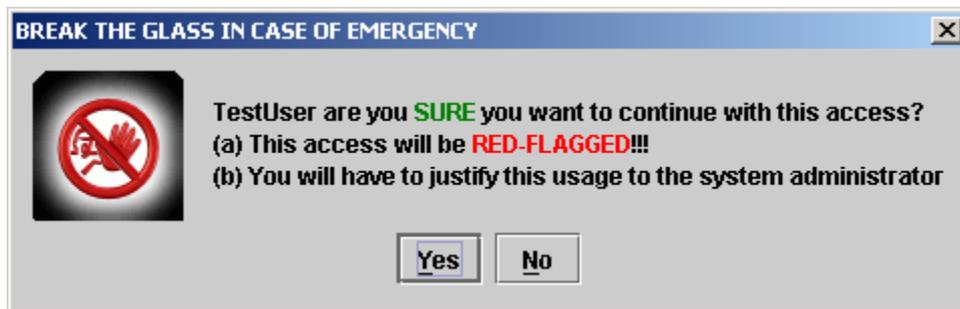
Step 6: Ongoing Obligations pop up while user accesses the file. The ongoing obligations window on the right is an example of an ongoing obligation that involves presenting the user with the security policies related to the file being accessed.



Step 7: Ongoing Conditions pop up intermittently if the ongoing conditions are not met whilst the user is accessing the file.

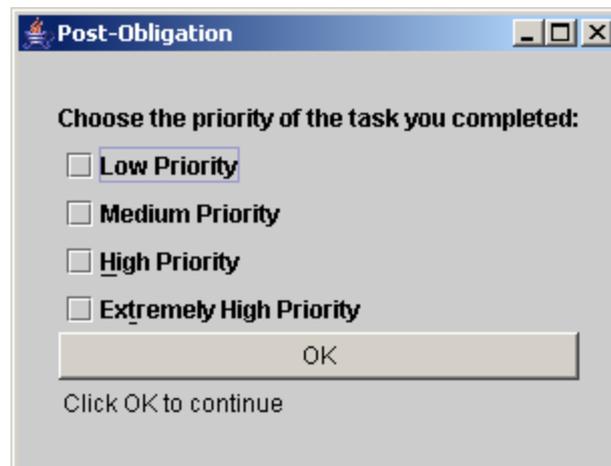


Step 8: The Break-the-Glass facility is invoked.

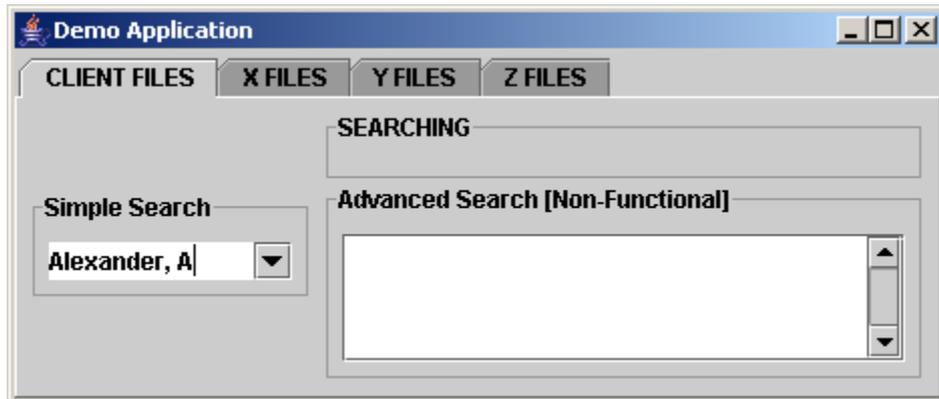


Step 9: User is allowed to sustain the access by accepting the Break-the-Glass option.

Step 10: Post-Obligations are invoked once the user closes the file concerned. This is an example of a post-obligation that is used to assess the user's trustworthiness.



Step 11: Back to Original Interface



8.6 Evaluation of the Aspect-Oriented Approach

In this section, the aspect-oriented approach is evaluated against the object-oriented approach. An object-oriented version of the proof-of-concept prototype was developed for comparative purposes.

8.6.1 The Design Approach

The UML diagram of the Object-Oriented Version is presented in Figure 8-4 below. Compared to the Aspect-Oriented Approach presented in Figure 8-3, it can be observed that the coupling between classes in the system have increased. It is not always clear how to best measure a metric such as coupling in an aspect-oriented system and how to compare it to its equivalent in a corresponding object-oriented system; as yet there does not exist a definitive work for metrics for aspect-oriented systems. However, it is still possible to observe fine-grained changes in coupling by reasoning about the changes in the code base (Singh, 2005).

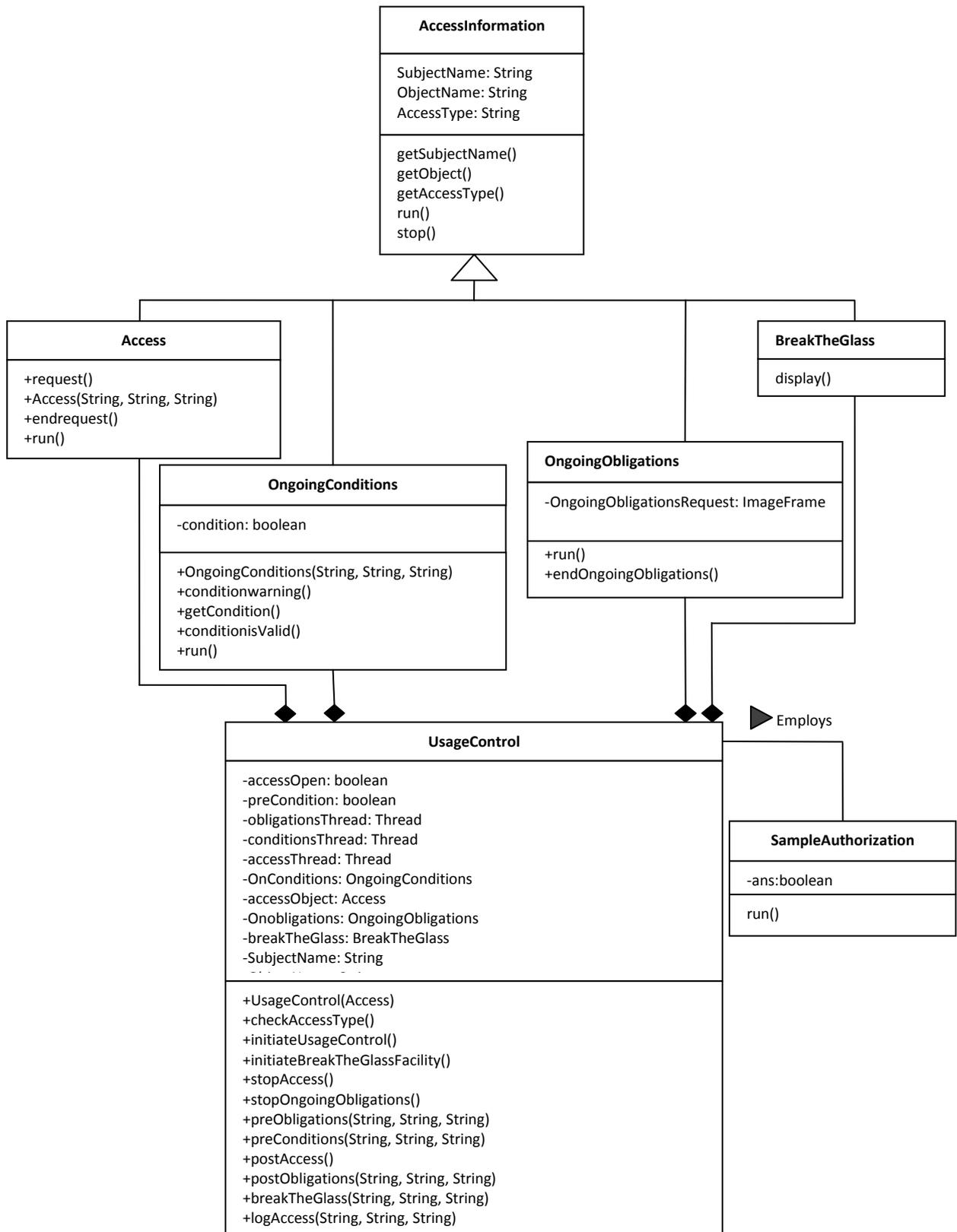


Figure 8-4: Showing the OOP UML of Core Classes

Compared to the aspect-oriented version, there appears to be a reduction in the scattering of concerns (see Figure 8-5) at the package level. For example, the aspect-oriented version does not cross-cut the class responsible for authorisation (i.e. `AuthorizationSim` in the diagram). In the case of the object-oriented version, the `Usagecontrol` class has to interact with the `Access`, `OngoingObligations` and `OngoingConditions` classes, as the class has to be aware of the fact that the conditions are no longer met; that the `OngoingObligations` are not being fulfilled; or that the user has terminated the request. In the case of the aspect-oriented version, the `UsageControlInjector` aspect observes each of these objects and decides what action to perform. It has been calculated that the usage control function is scattered across four classes in the object-oriented version.

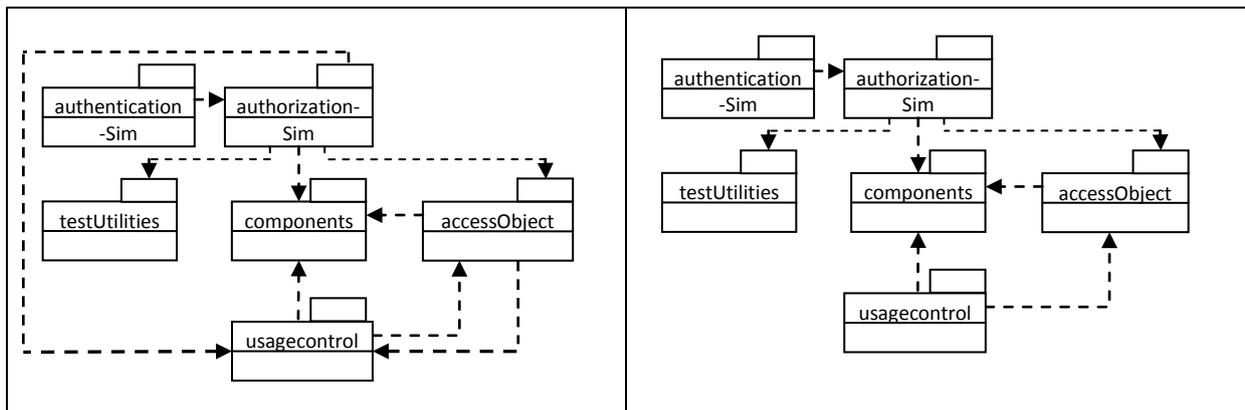


Figure 8-5: OOP package level diagram vs AOP package level diagram (on the right)

8.6.2 Execution Time and Memory Usage

Figure 8-6 shows the change in execution time. In the bar graph, the upper bar represents the time used by the aspect-oriented system while the lower bar represents the quantity for the object-oriented system. The values for these evaluations are calculated by averaging the data of several test runs. It can be noted that the object-oriented version is 1.9% faster than the aspect-oriented version, a difference that is actually negligible. Several reasons could account for this, such as user speed and the speed of the computer processor used for the experiments.

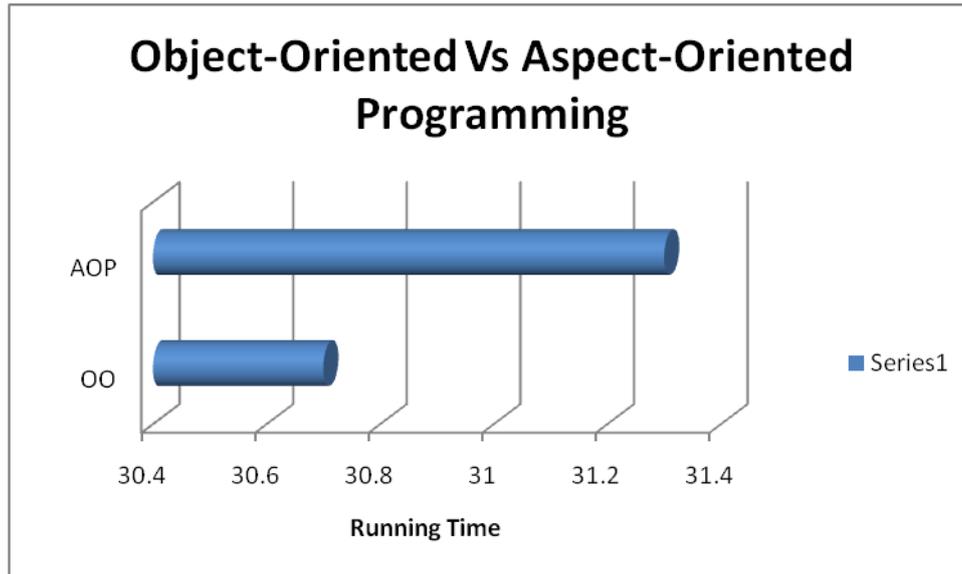


Figure 8-6: Showing comparisons of the execution time of OO vs AOP

Next, the amount of memory (Figure 8-7) that the Java Virtual Machine had demanded from the operating system at the end of each test run was compared. The object-oriented version used only 1.9% less memory than the aspect-oriented version. This figure is within a reasonable margin of error. The results of the tests, which were conducted on an Intel(R) Core 2 Duo CPU E6850 with 3.00 GHz and 1.96 GB of RAM, show that aspect-oriented programming can compete with object-oriented version.

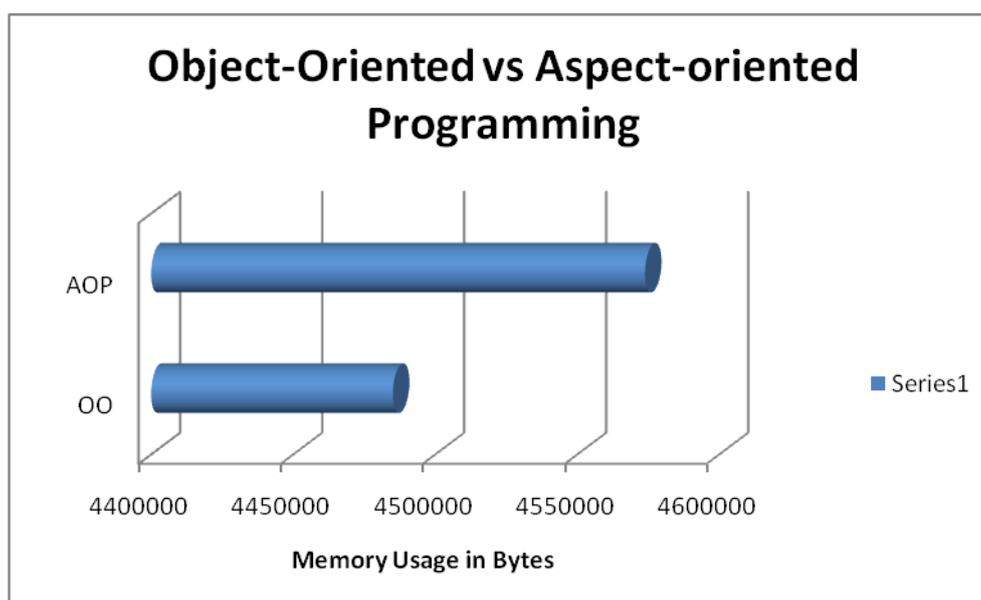


Figure 8-7: Showing comparisons of and Memory Usage of OO vs AOP

8.7 Evaluation of the model concept

The design science research methodology was used to conduct a small-scale experiment based on the following activities: build, evaluate, theorise, and justify (March and Smith, 1995). The experiment involved a problem identification stage (which was done in Chapter 1), design and development of prototypes stages, and an evaluation (Offerman et al., 2009) stage. The participants who were involved in the design and implementation of evaluative prototypes were Computer Science Honours students from the University of Pretoria. The concept specification was scaled up to a real-world scenario and included a mixed-initiative access control framework together with trust (see Appendix D). The purpose of this process was to identify if there were any vacuities, ambiguities or inconsistencies in the model concept. During the evaluation stage, the participants interacted with the evaluative prototypes and provided value judgements on it in terms of the efficacy of the security mechanism provided by the product concept.

A small segment of the evaluation involved the usability of the security mechanisms provided by the model concept. According to Jøsang and Patton (2001), security usability is concerned with the study of how security information should be handled in the user interface. In this context, the usability of the security mechanisms was evaluated. According to Whitten and Tygar (1999) security software is usable if the people who are expected to use it

- are reliably and made aware of the security tasks they need to perform;
- are able to figure out how to successfully perform those tasks;
- do not make dangerous errors; and
- are sufficiently comfortable with the interface to continue using it.

Qualitative data collections were employed, namely participant observation and open-ended interviewing (see Appendix D):

Observation: The idea with participant observation was to determine whether the end-user can successfully complete a task relating to the evaluative prototypes.

Qualitative interview: A qualitative, open-ended interview was conducted to determine the participants' perceptions of the appeal of the model concept in terms of data misuse.

Participants had to address the following in terms of the model concept: Weaknesses, Strengths, Potential improvements, Viability, Applicability and Scalability.

To facilitate the process, the issues concerning the evaluation were formulated into 11 statements. The participants then provided a judgment on each statement. The following data was gathered from the experiment and are discussed in the paragraphs that follow:

Statement 1: The product specifications as given in the assignment were ambiguous and incomplete.

Statement 2: The product specifications as given in the assignment could easily be translated into an implementable product.

Both statements above focused on the viability (or not) of the product, and it was found that 78% of the participants judged the specifications to be unambiguous and complete. Two participants stated that the notion of priorities of tasks needed to be addressed, as the priorities of tasks were assigned randomly in the specification. The priority of the task and the use of the Break-the-Glass feature were examined to determine whether the user utilised the Break-the-Glass facility for a bona fide emergency. If this priority of task did not warrant 'breaking the glass', then the user's rights to information under the optimistic access control domain were constrained. Three of the participants in the study felt that the specifications given were ambiguous and incomplete, and claimed that this had led to misinterpretation. Another participant claimed that the specification "did not give explicit rules for the break-the-glass". All participants nevertheless agreed that the specifications could easily be translated into an implementable product. In fact, one participant indicated that the specifications were easy to divide into implementable components. The product concept was judged to be highly viable and the participants were able to implement it using several approaches, including Java, C# and PHP.

Statement 3: In terms of the enforcement of security, other mechanisms such as a written policy document or adequate training would have been more effective than the mechanisms identified in the product concept.

With regard to the effectiveness of the product concept in relation to other non-technical approaches, 78,5% of the participants disagreed that other mechanisms such as training would have been more effective than the product concept. Three participants felt that other mechanisms – in combination – would increase the security overall, while four others felt that the training and policy documents were "simpler to ignore" and "not a constant reminder" as was the case with an automated system. In addition, the prototype concept enabled the tracking of a user's actions.

Statement 4: The flexibility offered under the optimistic access control domain is a security risk.

Based on the risk of using optimistic access control (owing to its flexibility), 78,5% agreed that optimistic access control was a security risk and that data should be protected by other means. However, some participants indicated that it depended on the nature of the organisation and its data, and that some environments such as the medical industry actually required the proposed level of flexibility.

Statement 5: Specifying system conditions, such as limiting access according to the time-of-day, may deter users from abusing their privileges.

Altogether 78.5% of the participants agreed that specifying conditions would deter users from abusing their privileges. Most participants felt that these conditions would give the user the feeling that they were "doing something wrong" and that they would be deterred as a result. They also felt that the threat of punishment and losing trust might provide a motivation for users not to abuse their privileges.

Statement 6: The 'break-the-glass' facility is vulnerable to abuse.

In terms of the susceptibility of the break the glass facility 71% of the participants agreed that the Break-the-Glass system was vulnerable to abuse. However, most of them indicated that the threat of being discovered after the event was a way of preventing the Break-the-Glass facility from being misused.

Statement 7: The protection mechanisms, such as fulfilling obligations, will compel users to comply with the established rules of behaviour in order to protect confidential information.

In terms of satisfying obligations, 85% of the participants agreed that the fulfilment of obligations would compel users to comply with the established rules of behaviour. Using obligations would prevent users from claiming ignorance as an excuse for not complying. Furthermore, since users are intimidated by warnings, user responsibility could be expected to increase.

Statement 8: An individual who interacts with the system will recognize that access is dependent on user responsibility as well as technical access control.

With regard to security usability, 71.4 % of the participants agreed that an individual who interacted with the system would recognise that access was dependent on user responsibility as well as technical access control. Those participants who opposed the statement argued that users were irresponsible and untrustworthy.

Statement 9: The risk of losing one's rights to information under the optimistic access control domain may deter one from abusing one's privileges.

Due to the severity of punishment, 84.6% agreed that the risk of losing one's rights to information under the optimistic access control domain might act as a deterrent against abusing one's privileges. The threat of being caught and losing one's trust was a strong motivator. However, participants agreed that if the user's premeditated goal was to steal data, these mechanisms would not prevent such incidences.

Statement 10: The conditions, obligations and the break-the-glass mechanisms may be distracting to a user.

In terms of security usability, 57% of the participants disagreed that the conditions, obligations and the Break-the-Glass mechanism would be distracting to the user. Even those users who agreed felt that after some time most users would ignore these pop-ups anyway. However, all of this would depend on how the user interface was designed. It was important to be presented in such a way that users should not become complacent or exasperated about the messages.

Statement 11: Most users will ignore the messages about conditions and obligations relating to the access.

Half of the participants agreed that users would ignore the messages about conditions and obligations relating to the access. They felt that, in time, users would eventually pay no attention to these messages. The other 50% of participants, who disagreed, proposed that users should be forced to respond to the message. Furthermore, participants posited that users would ignore these messages unless the consequences were clearly specified.

Although most participants regarded optimistic access control as a security risk, participants reasoned that the additional facilities of obligations and conditions might deter users from abusing their privileges. Participants suggested that constant reminders would ensure that users would not perform illegitimate actions seeing that they would be monitored. Some participants indicated that the separation of public domain information from the private domain was strength, as it allowed for information to be subject to different controls. Furthermore, it was quite a simple task to maintain the access control policies for information under the optimistic access control domain. Regarding improvements, it was suggested that the conditions should be more dynamic and that they should be based on user profiles. Participants also suggested that rather than displaying pop-ups for every access, a single-sign or a pop-up should be flashed intermittently. Regarding weaknesses, the participants felt that users might try to bypass warning messages because they were annoyed by them and that there was too much reliance on the trustworthiness of users.

The evaluation exercise revealed that the model concept could be appropriate to call centres, dynamic environments, medical information systems and Wikipedia. It was reasoned that the system would be relevant in situations where users were transitory. This kind of system would also be more fitting for users who were professionals rather than the average user. It would furthermore be more suitable in environments where damage was reversible or in small organisations that used data that was not that sensitive.

Augmenting traditional access control with usage control features is expected to slow down program execution, as it involves the inclusion of additional code in the functional system. In terms of security usability, controls such as pre-obligations and ongoing obligations may be distracting and impact negatively on the productivity of users. Perhaps, as the user becomes more 'trustworthy', some obligations or conditions may be relaxed or negotiated. The costs of implementing usage control as a deterrent may have to be weighed up against the cost of information misuse. South Africa's draft bill on the protection of personal information is viewed as a means to ensure South Africa's future participation in the information market by providing 'adequate' information protection of an international standard (see (*CHAPTER 9: A DRAFT BILL ON THE PROTECTION OF PERSONAL INFORMATION, 2005*)). If individuals are ensured that their privacy is taken into account in a software system, it is understandable that they will trust the system with their private information. The survival of e-business will probably depend on its ability to guarantee the privacy of its clients.

The proposed OAC(UCON) model does not account for trust issues; thus this needs to be addressed in future renditions of the model. In addition, the relationship between using the Break-the-Glass facility and the priority of the task needs to be explored. Since using this facility is dependent on the urgency of the task, the rules governing the Break-the-Glass facility need to be defined in more detail.

8.8 Conclusion

The aspect designed for the enhancement of optimistic access control was tested in terms of a proof-of-concept prototype. It was found that confining all the operations relating to usage control to a single modular structure would reduce both development and maintenance costs. Next, the relationship between multithreading and cross-cutting behaviour was explored in this chapter. It was shown that the aspect-oriented approach does not impact significantly on execution time or memory usage and that aspect-oriented programming introduced fewer scattering of usage control concerns and less coupling between classes.

Owing to the sample size which was quite modest (i.e. 14 participants), the limitations of the experiment need to be taken into account when making generalisations from the research. The nature of the study required participants to be competent at programming a large system independently and to deliver the product within a reasonable time frame. Purposive sampling had to be employed as the participant had to be an advanced programmer who also had the time available to do the task. These two requirements were met by the students enrolled for the Computer Science Honours programme at the University of Pretoria. It is difficult to find members of society who would fit this unique profile. The other limitation posed by the research method was that every participant's final product had to be evaluated and the participants had to share their insights on the model as well as their design decisions. Having a large sample would make this task extremely time consuming. It would also imply that each participant's involvement would have been superficial. A small sample, on the other hand, allowed for a more in-depth analysis of each participant's value judgement. A future study may involve replicating the research method with a new group of students from another university.

The model has not been tested within a large distributed system with several end-users in an organisational setting. However, participants who tested the model concept can be considered the representatives of stakeholders in the information technology industry. As postgraduate students, they have extensive knowledge of information systems and are



currently employable or employed within the information systems sector. The product was found to be highly viable as all participants were able to implement the scaled-up version of the concept. The usability of the system was reasonable, except for the criticism that the usage control features might be distracting and could eventually be ignored. However, the evaluation revealed that users would understand that access control was based on technical control as well as user responsibility. The effectiveness of optimistic access control was found to be largely dependent on the usage control features, as optimistic access control on its own posed a security risk. The evaluation nevertheless proved that by employing usage control features of obligations and conditions, this risk would be reduced.

CHAPTER 9:

CONCLUSION

9.1 Introduction

The study in hand focused on a model for usage control under the optimistic access control paradigm, i.e. the OAC(UCON) model. To increase the applicability of the model, it was presented within a mixed-initiative access control framework. The pragmatic issue of implementing such a model within the wider context of access control formed the topic of discussion in this thesis. To ease the integration of the proposed model into an existing access control framework, an aspect-oriented approach was selected. The motivation for this study was posited in Chapter 1 and required a number of research goals to be addressed. In this closing chapter the researcher evaluates the extent to which the objectives of the research goals have been met. Finally, it concludes with a discussion of the main contribution of the research and suggestions for further research.

9.2 Main contribution

This research did not promote the notion that traditional access control models were inferior to optimistic access control. Rather, it suggested that the two approaches might work well in a mixed-initiative approach. The OAC(UCON) model is flexible and reduces the burden of setting pre-configured security policies for every subject-object relationship, and thereby reduces the load on system administrators. However, the model acknowledges that the gains realised by flexibility should not be negated through data misuse. Thus, the model provided sufficient deterrents against data misuse by leveraging the security mechanisms

offered by usage control. It was suggested that data that cannot be reasonably protected within traditional access control could be protected by these usage control deterrents.

As was stated earlier, the proposed solution could well ease the burden of system administrators significantly. It is rather difficult for administrators to predict all of the possible usage scenarios and thus all of the necessary permissions. With optimistic access control, it is ultimately left to the users to make that judgement. Consequently, the complexity of implementing and maintaining pre-configured access control policies is shifted to the way the user interacts with the system. Adapting usage control as a deterrent provides a proactive mechanism over and above the retroactive methods of auditing and accountability. By using the OAC(UCON) model, a larger subset of information may be relegated into the public domain.

This research also addresses the issue of continuity within usage control and its practical implementation within the access control context rather than within the digital rights management context. The thesis is consequently presenting pragmatic ways of introducing continuity within the access control dimension. In terms of the proof-of-concept that was developed, the ongoing obligations involved presenting the user with the relevant security policies while he/she accesses the related information. This is an example of the type of application that educates the user on approved security policies with regard to the specific data that he/she is interacting with.

Investigating the efficacy of the aspect-oriented programming language can be considered one of the major contributions of this research. It was found that usage control can be completely separated from access control and other application logic. It was also determined that the performance differences between the object-oriented and aspect-oriented version were negligible. Additionally, there was less coupling between classes with the aspect-oriented version, which increased the readability and understandability of the code. The relationship between multithreading and cross-cutting behaviour was also explored and the study demonstrated how ongoing authorisations could be maintained with multithreading.

The model is unique in that the access controls are applied in the application layer. It provides supplemental usage control to objects that have their access rights defined within the database layer. The rights defined in the database layer may be relaxed in the application layer or maintained as specified. If the rights were relegated to optimistic rights, then the rights are relaxed and supplemented with optimistic rights. Alternatively, if these rights were considered to be highly classified, then the OAC(UCON) allowed these rights to remain as specified.

9.3 Revisiting the problem statement

The problem statement highlighted the inadequacies of current access control models, namely their lack of flexibility and difficulties in assigning pre-configured access control policies. To this end, a critical overview of popular access control models was provided and an optimistic access control model was recommended as a means of correcting these deficiencies. Since it was noted that optimistic access control is far too flexible to be used in practice, it was enhanced with usage control in order to offer greater rigour. In this model, usage control was reformulated under optimistic access control to act as a mechanism for deterrence rather for denial of access. Thus the OAC(UCON) model was developed. To improve its general applicability, it was presented in a mixed-initiative access control framework, where pessimistic access control models were complemented with optimistic access control models. In order for this type of integration to be successful, a software approach was inferred that would allow for the seamless augmentation of traditional access control with optimistic access control enhanced with usage control, namely the aspect-oriented approach. A partially generic usage control aspect was presented that could, in theory and with minor modifications, be augmented seamlessly into a fully operational system. The aspect-oriented approach was also evaluated in terms of performance against an object-oriented approach. Finally, the design science research methodology was employed to test the model concept and to assess its scalability with other access control measures and within a wider context so as to gain insight into the usability of the model concept.

9.4 Future Research Directions

The element of trust within the OAC(UCON) model warrants an investigation into human behaviours and the responses to its application. It would be pragmatic to investigate whether the model concept in fact dissuades individuals from accessing and misusing information in the public domain. Future research could be directed at the inclusion of trust-based mechanisms to update a user's optimistic rights. Presently the model does not account for how trust levels may change when a user loses his/her optimistic rights. In order to test the scalability of the model concept, the notion of trust needs to be considered and its inclusion would complete the mixed-initiative access control framework. The issue of trust was considered in terms of how a user's rights to information under the optimistic paradigm may be modified based on prior usage. In the case of the evaluative prototype of the model it was presumed that, at the onset, each user had access to optimistic rights rated as 'high'. However, as the user demonstrated his/her untrustworthiness, the level of access was downgraded to 'medium' and finally to 'low'. As their optimistic rights were demoted, the view to information became increasingly constrained. The users' optimistic rights were updated using fuzzy logic. Future research could well involve considering the factors that influence trust levels. In the specification given to the participants as part of the design research method, the priority of the task and the user's previous trust level were used to update his/her optimistic rights in a fuzzy matrix.

An alternative research direction may involve investigating whether the model concept increases the propensity towards compliant information security behaviour. This refers to a set of core information security activities that has to be carried out by end-users so as to maintain information security as defined by information security policies (Chan et al., 2006).

It is also suggested that future studies should involve a case study to test the usability of the aspect-oriented approach since it has not been tested in an organisational context yet. However, confining all the operations pertaining to usage control to a single modular structure will alleviate both development and maintenance costs as it can be integrated seamlessly into a system based on traditional access control.

9.5 Conclusion

The proposed solution to access control draws inspiration from some of the principles advocated by agile methods. For example, consider the agile principles relating to embracing change and maintaining simplicity. In the case in hand access control was implemented in its most rudimentary form. As with agile methods, the reliance was on people rather than on complicated processes to maintain control.

The viability of the model concept was demonstrated in a scaled-up version where it was possible to create a mixed-initiative access control model. It was found that optimistic access control is a security risk, but that the combination of usage control features coupled with monitoring and punitive action may deter users from abusing their privileges. The security usability aspect of the concept would need to be improved, as users would probably sooner or later disregard the obligations and conditions. Accordingly these notions needed to be more dynamic and responsive. The obligations and conditions messages need to be updated constantly and they have to be reformulated to retain a user's focus. The evaluation revealed that the concept may be appropriate to call centres, medical information systems, temporal environments and smaller organisations where data is not viewed as particularly sensitive. This kind of system would also be more appropriate for users who have a degree of professionalism more so than the average user and in environments where damage is reversible.

The use of aspect-oriented programming contributed to the principles of embracing change and maintaining simplicity. Adapting usage control as a deterrent provides a proactive mechanism over and above the retroactive methods of auditing, accountability and recoverability. It is envisioned that a larger subset of information may be transferred to the public domain, thus obviating the need for specifying convoluted access control policy decisions.

REFERENCES

Alexander, R.T. & Bieman, J.M. (2002) 'Challenges with Aspect-oriented Technology', ICSE Workshop on Software Quality, Orlando, Florida, 25 May 2002.

Anderson, R.J. (2001) *Security engineering: a guide to building dependable distributed systems*, Wiley, Computer Publishing, New York, USA.

Andrews, G.R. & Reitman, R.P. (1980) 'An axiomatic approach to information flow in programs', *ACM Transactions on Programming Languages and Systems*, vol. 2, no. 1, pp. 56-76.

Baniassad, E. & Clarke, S. (2004) ' Finding Aspects in Requirements with Theme/Doc', in *Proceedings of Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design*, Lancaster, UK, 22 March 2004.

Bell, D.E. & La Padula, L.J. (1976) *Secure computer systems: Unified Exposition and Multics Interpretation*, Technical Report ESD-TR-75-306, Electronics Systems Division, Bedford USAF Base.

Bodkin, R. (2004) 'Enterprise Security Aspects', AOSD'04 International Conference on Aspect-Oriented Software Development, Lancaster, UK, March 2004, <[Online] Available: <http://www.cs.kuleuven.ac.be/~distrinet/events/aosdsec/papers.html>>.

Boehm, B. (2002) 'Get Ready for Agile Methods with Care', *Computer*, vol. 35, no. 1, pp. 64-9.

Boström, G. (2004) 'A case study on estimating the software engineering properties of implementing Database Encryption as an aspect', Proceedings of the 3rd international conference on Aspect-oriented software development, Lancaster, UK, 22-24 March 2004.

Briscoe, B., Rizzo, M., Tassel, J. & Damianakis, K. (2000) 'Lightweight policing and charging for packet networks', in *3rd IEEE Conference on Open Architectures and Network Programming*, Tel Aviv, Israel, 26-27 March 2000, pp. 77-87.

Cederquist, J.G., Corin, R., Dekker, M.A.C., Etalle, S., den Hartog, J.I. & Lenzeni, G. (2006) *The Audit Logic: Policy Compliance in Distributed Systems*, Centre for Telematics and Information Technology, University of Twente, Enschede.

Chan, M., Woon, I. & Kankanhalli, A. (2006) 'Perceptions of information security in the workplace: linking information security climate to compliant behavior', *Journal of Information Privacy and Security*, vol. 1, no. 3, pp. 18-41.

CHAPTER 9: A DRAFT BILL ON THE PROTECTION OF PERSONAL INFORMATION (2005), viewed 30 November 2009, <Available [Online] <http://old.ispa.org.za/regcom/privacyfiles/chapter-9-draft-bill-protection-personal-info.pdf>>.

Chen, L. (2004) *Aspect-Oriented Programming in Software Engineering*, Technical Report, Wake Forest University, Department of Computer Science.

Chon, R., Enokido, T. & Wietrzsk, V. (2004) 'Role Locks to Prevent Illegal Information Flow among Objects', in *18th International Conference on Advanced Information Networking and Applications (AINA'04) Volume 1*, Fukuoka, Japan, 29-31 March 2004, pp. 196-201.

Chou, S.-C. (2003) 'Information Flow Control among Objects: Taking Foreign Objects into Control', in *36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, Big Island, Hawaii, 6-9 January 2003, pp. 335-44.

Clarke, R.E. (2002) 'e-Consent: A Critical Element of Trust in e-Business', 15th Bled Electronic Commerce Conference: eReality:Constructing the eEconomy, Bled, Slovenia, 17-19 June 2002.

Constantinides, C. & Hasson, Y. (2002) 'Beyond objects: Improving the modularity of complex software', Workshop on Grand Challenges for Computing Research, Edinburgh, Scotland, 24-26 November 2002.

D' Arcy, D. & Hovav, A. (2007) 'Deterring internal information systems misuse', *Communications of the ACM*, vol. 50, no. 20, pp. 113-7.

De Win, B., Joosen, W. & Piessens, F. (2002) 'Developing Secure Applications through Aspect-Oriented Programming', in Aksit, M., Clarke, S., Elrad, T. & Filman, R.E. (eds), *Aspect-Oriented Software Development*, Addison-Wesley, Boston, p. 633–50.

De Win, B., Joosen, W. & Piessens, F. (2003) 'AOSD & Security: A Practical Assessment', Workshop on Software engineering Properties of Languages for Aspect Technologies (SPLAT03), Boston, Massachusetts, 17-21 March 2003.

De Win, B., Piessens, F. & Joosen, W. (2002) 'On the importance of the separation-of-concerns principle in secure software engineering', Workshop on the Application of Engineering Principles to System Security Design, Boston, Massachusetts, 6-8 November 2002.

De Win, B., Vanhaute, B. & De Decker, B. (2002) 'How aspect-oriented programming can help to build secure software', *Informatica*, vol. 26, no. 2, pp. 141-9.

De Win, B., Vanhaute, B. & Decker, B. (2001) 'Security Through Aspect-Oriented Programming', in Decker, B.D., Piessens, F., Smits, J. & Herreweghen, E.V. (eds), *Advances in*

Network and Distributed Systems Security, IFIP TC11 WG11.4 First Working Conference on Network Security, Leuven, Belgium, 26-27 November 2001, pp. 125-38.

Denning, D.E. & Denning, P.J. (1977) 'Certification of Programs for Secure Information Flow', *Communications of the ACM*, vol. 20, no. 7, pp. 504 -13.

Devanbu, P.T. & Stubblebine, S. (2000) 'Software engineering for security: a roadmap', in *Proceedings of the Conference on The Future of Software Engineering*, Limerick, Ireland, 4-11 June 2000, pp. 227-39.

Dewan, P., Grundin, J. & Horvitz, E. (2007) 'Towards a mixed-initiative access control', in *COLCOM '07: Proceedings of the 2007 International Conference on Collaborative Computing: Networking, Applications and Worksharing*, New York, USA, November 12-15, 2007, pp. 64-71.

Downs, D., Rub, J.R., Kung, K.C. & Jordan, C.S. (1985) 'Issues in Discretionary Access Control', in *1985 IEEE Symposium on Security and Privacy, 1985*, Oakland, CA, 22-24 April 1985, pp. 208-15.

Elrad, T.M., Askit, G., Kiczales, K., Lieberherr, H. & Ossher. (2001) 'Discussing Aspects of AAOP', *Communications of the ACM*, vol. 44, no. 10, pp. 33-8.

Engel, M. & Freisleben, B. (2005) 'Supporting autonomic computing functionality via dynamic operating system kernel aspects', in *Proceedings of the 4th international conference on Aspect-oriented software development*, Chicago, Illinois, 22-26 March 2005, p. 51 – 62.

English, C., Nixon, P., Terzis, S., McGettrick, A. & Lowe, H. (2002) 'Security Models for Trusting Network Appliances', 5th Annual Workshop on Networked Appliances, Liverpool, England, October 2002.

Esquivel, A., Haya, P.A. & García-Herranz, M. (2007) 'Managing Pervasive Environment Privacy Using the "fair trade" Metaphor', in Meersman, R., Tari, Z. & Herrero, P. (eds), *On the move to meaningful Internet systems: OTM 2007 Workshops*, Springer-Verlag, Berlin, Germany, vol. 4806, Lecture Notes in Computer Science, pp. 804-13.

Etalle, S. & Winsborough, W.H. (2007) 'A Posteriori Compliance Control', in *SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies*, Sophia Antipolis, France, 20-22 June 2007, pp. 11-20.

Falcarin, P., Baldi, M. & Mazzocchi, D. (2004) 'Software Tampering Detection using AOP and mobile code', 3rd International Conference on Aspect-Oriented Software Development (AOSD'04), Lancaster, UK, 22-24 March 2004.

Ferreira, A., Cruz-Correia, R., Antunes, L., Farinha, P., Oliveira-Palhares, E., Chadwick, D.W. & Costa-Pereira, A. (2006) 'How to break access control in a controlled manner', in *Proceedings of the 19th IEEE International Symposium on Computer-Based Medical Systems*, Salt Lake City, Utah, 22-23 June 2006, pp. 847-51.

Georgiev, I.K. & Georgiev, I.I. (2001) 'A security model for distributed computing', *Journal of computing sciences in colleges*, vol. 17, no. 1, pp. 178-86.

Grandison, T.W.A. (2003) 'Trust Management for Internet Applications', PHD thesis, Imperial College London.

Groher, I. & Schulze, S. (2003) 'Generating Aspect Code from UML Models', Workshop on Aspect-Oriented Modeling with UML, AOSD, Boston, Mass. USA, 17-21 March 2003.

Grundy, J. & Ding, G. (2002) 'Automatic Validation of Deployed J2EE Components Using Aspects', in *17th IEEE International Conference on Automated Software Engineering (ASE'02)*, Edinburgh, UK, 23-27 September 2002, pp. 47-57.

Haldar, V., Chandra, D. & Franz, M. (2005) 'Practical, Dynamic Information Flow for Virtual Machines', in *PLID'05 2nd International Workshop on Programming Language Interference and Dependence*, London, UK, 6 September 2005.

Harrison, W. & Ossher, H. (1993) 'Subject-oriented programming: a critique of pure objects', in *Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications*, Washington D.C., 26 September – 1 October, pp. 411-28.

Herath, T. & Rao, H.R. (2009) 'Protection motivation and deterrence: a framework for security policy compliance in organisations', *European Journal of Information Systems*, vol. 18, no. 2, pp. 106-25.

Higgins, G.E., Wilson, A.L. & Fell, B.D. (2005) 'An Application of Deterrence Theory to Software Piracy', *Journal of Criminal Justice and Popular Culture*, vol. 12, no. 3, pp. 166-84.

Hong, J.I. & Landay, J.A. (2004) 'An Architecture for Privacy Sensitive Ubiquitous Computing', in *Proceedings of the International Conference on Mobile Systems, Applications and Services*, Boston, Massachusetts, USA, pp. 177-89.

Houmb, S.H., Georg, G., France, R. & Matheson, D. (2004) 'Using aspects to manage security risks in risk-driven development', in *3rd International Workshop on Critical Systems Development with UML*, Lisbon, Portugal, 11-15 October, pp. 71-84.

Imine, A., Cherif, A. & Rusinowitch, M. (2009) *An Optimistic Mandatory Access Control Model for Distributed Collaborative Editors*, Technical Report, INRIA.

Izaki, K., anaka, K. & Takizawa, M. (2001) 'Information Flow Control in Role-Based Model for Distributed Objects', in *Eighth International Conference on Parallel and Distributed Systems*, Kyongju City, Korea, 26-29 June 2001, pp. 363-70.

Jones, R.L. & Rastogi, A. (2004) 'Secure Code: Building Security into the Software Development Life Cycle', *Information Security Journal: A Global Perspective*, vol. 15, no. 5, pp. 29-39.

Jøsang, A. & Patton, M. (2001) *User Interface Requirements for Authentication of Communication*, Technical Report, Distributed Systems Technology Centre, Brisbane, Australia.

Kersten, M. (2005) *AOP Tools Comparison, AOP@Work, DeveloperWorks, IBM*, viewed 1 December 2005, <[Online] Available: <http://www.ibm.com/developerworks/library/i-aopwork1>>.

Kiczales, G. (1996) 'Aspect-Oriented Programming', *Computing Surveys(CSUR)*, vol. 28, no. 4, p. 154.

Kiczales, G., Hillsdale, E., Hugunin, J., Kersten, M., Palm, J. & Griswold, W.G. (2001) 'Getting Started with AspectJ', *Communications of the ACM*, vol. 44, no. 10, pp. 59-65.

Kiczales, G., Irwin, J., Lamping, J., Loingtier, J., Lopes, C.V., Maeda, C. & Mendhekar, A. (1997) 'Aspect-Oriented Programming', in Aksit, M. & Matsuoka, S. (eds), *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP)*, Springer-Verlag, Jyväskylä, Finland, vol. 1241, Lecture Notes in Computer Science, pp. 220-40.

Kim, S. & Leem, C.S. (2004) 'An Information Engineering Methodology for the Security Strategy Planning', in Lagana, A. (ed.), *Computational Science and Its Applications - ICCSA 2004*, Springer, Berlin/Heidelberg, vol. 3043, Lecture Notes in Computer Science, pp. 597-607.

Kumar, A., Singh, A.K. & Babu, R.S. (2001) 'A security assurance framework for component based software development', *Informatica*, vol. 25, no. 4, pp. 509 - 15.

Läufer, K., Thiruvathukal, G.K., Elrad, T. & Bader, A. (2003) 'Enhancing the CS Curriculum with Aspect Oriented Software Development (AOSD), Working Paper', International Conference on aspect-oriented software development, Boston, 17-21 March 2003.

Lee, G., Kim, W. & Kim, D.-K. (2004) 'Novel Method to Support User's Consent in Usage Control for Stable Trust in E-business', in Lagan, A., Gavrilova, M.L., Kumar, V., Mun, Y., Tan, C.J.K. & Gervasi, O. (eds), *Computational science and its applications - ICCSA 2004*, vol. 3045, Lecture Notes in Computer Science, pp. 906-14.

Li, N., Moa, Z. & Chen, H. (2009) 'Usable Mandatory Access Control for Operating Systems', in Roa, H.R. & Upadhyaya (eds), *Information Assurance, Security and Privacy (Handbooks in Information Systems)*, Emerald Group Publishing Limited, Bingley, UK, vol. 14, pp. 335-63.

Li, X., Naeem, N.A. & Kemme, B. (2005) 'Fine-Granularity Access Control in 3-tier Laboratory Information Systems', in *Proceeding of the 9th Database Engineering and Application Symposium, (IDEAS '05)*, Montreal, Canada, 25-27 July 2005, pp. 391- 7.

Mao, Z., Li, N., Chen, H. & Jiang, X. (2009) 'Trojan horse resistant discretionary access control', in *SACMAT '09: Proceedings of the 14th ACM symposium on Access control models and technologies*, Stresa, Italy, pp. 237-46.

March, M.T. & Smith, G.F. (1995) 'Design and natural science research on information technology ', *Decision Support Systems*, vol. 15, no. 4, pp. 251-66.

Masuhara, H. & Kawauchi, K. (2003) 'Dataflow Pointcut in Aspect-Oriented Programming', in Ohori, A. (ed.), *Proceedings of The First Asian Symposium on Programming Languages and Systems (APLAS'03)*, Springer, Beijing, China, vol. 2895, Lecture Notes in Computer Science, pp. 105-21.

McCollum, C.J. & Messing, J.R.N.L. (1990) 'Beyond the Pale of MAC and DAC Defining new forms of access control', in *Proceedings of IEEE Symposium on Security and Privacy*, Oakland, California, USA, 17-19 May 1990, pp. 190-200.

Miller, S.K. (2001) 'Aspect-Oriented Programming Takes Aim at Software Complexity', *Computer*, vol. 34, no. 4, pp. 18-21.

Murphy, G.C., Walker, R.J. & Baniassad, E.L.A. (1999) 'Evaluating Emerging Software development Technologies: Lessons learned from Assessing Aspect-Oriented Programming', *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 483-55.

Murphy, G.C., Walker, R.J., Baniassad, E.L.A., Rollibard, M.P., Lai, A. & A., K.M. (2001) 'Does aspect-oriented programming work?' *Communications of the ACM*, vol. 44, no. 10, pp. 75-7.

Offerman, P., Levina, O., Schonherr, M. & Bub (2009) 'Outline of a design science research process', in *4th International Conference on Design Science Research into Systems and Technology*, Malvern, Pennsylvania, 6-9 May 2009.

Osborn, S., Sandhu, R. & Munawer, Q. (2000) 'Configuring Role-Based Access Control to Enforce Mandatory and Discretionary', *ACM Transactions on Information and System Security*, vol. 3, no. 2, pp. 85-106.

Padayachee, K. (2007) 'Instrumentation of AspectJ Programs: An Exploratory Study', in *Proceedings of the International MultiConference of Engineers and Computer Scientists 2007*, Hong Kong, vol. 1, 21- 23 March 2007, pp. 1077-81.

Padayachee, K. & Eloff, J.H.P. (2006) 'The Next Challenge: Aspect-Oriented Programming', in Nyongesa, H. (ed.), *Proceedings of the Sixth IASTED International Conference on Modelling, Simulation and Optimization*, Gaborone, Botswana, 11-13 September 2006, pp. 304-7.

Padayachee, K. & Eloff, J.H.P. (2007) 'Enhancing Optimistic Access Controls with Usage Control', in Lambrinouidakis, C., Pernul, G. & Tjoa, A.M. (eds), *Trust, Privacy and Security in Digital Business*, Springer, Regensburg, Germany, vol. 4657, Lecture Notes in Computer Science, pp. 75 - 82.

Padayachee, K. & Eloff, J.H.P. (2009) 'Adapting usage control as a deterrent to address the inadequacies of access controls', *Computers and Security*, vol. 28, no. 7, pp. 536-44.

Padayachee, K. & Wakaba, N. (2007) 'A Taxonomy of Aspect-Oriented Security', The 2007 European Applied Business Research Conference, Venice, Italy, 4-7 June 2006.

Park, J., Zhang, X. & Sandhu, R. (2004) 'Attribute Mutability in Usage Control', in *Proceedings of the annual IFIP WG 11.3 Working Conference on Data and Applications Security*, Sitges, Catalonia, Spain, 26 July 2004, pp. 15-29.

Pavlich-Mariscal, J., Michel, L. & Demurjian, S. (2005) 'A Formal Enforcement Framework for Role-Based Access Control using Aspect-Oriented Programming', in *Proceedings of ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML 2005)*, Montego Bay, Jamaica, 2-7 October 2005, pp. 537-52.

Pfleeger, C.P. (1997) *Security in Computing*, 2nd edn, Engelwood Cliffs, NJ.:Prentice Hall, United States of America.

Pfleeger, C.P. & Pfleeger, S.L. (2003) *Security in Computing*, 3rd edn, Prentice Hall, Upper Saddle River, New Jersey.

Pieprzyk, J., Hardjono, T. & Seberry, J. (2003) *Fundamentals of computer security*, Springer, Berlin.

Pohl, C., Charfi, A., Gilani, W., Göbel, S. & B.G., H. (2008) 'Adopting Aspect-Oriented Software Development in Business Application Engineering', 7th International Conference on Aspect-Oriented Development, Brussels, Belgium, 31 March - 4 April 2008.

Povey, D. (1999) 'Optimistic Security: A New Access Control Paradigm', Proceedings of the 1999 workshop on New security paradigms, Caledon Hills, Ontario, Canada, 22 - 24 September 1999.

Pretschner, A., Hilty, M., Schutz, F., Schaefer, C. & Walter, T. (2008) 'Usage Control Enforcement: Present and Future', *IEEE Security & Privacy*, vol. 6, no. 4, pp. 44-53.

Pretschner, A. & Walter, T. (2008) 'Negotiation of Usage Control Policies - Simply the Best?' in *ARES '08: Proceedings of the 2008 Third International Conference on Availability, Reliability and Security*, Washington, DC, USA, 4-7 March 2008, pp. 1135-6.

Pudney, P. (2003) *e-Consent in consumer health & telemedicine*, University of South Australia, viewed 30 November 2009, <[Online] Available: <http://www.pudney.net.au/~phillip/papers/econsent.pdf>>.

Raje, R.R., Zhong, M. & Wang, T. (2001) 'Case Study: A Distributed Concurrent System with AspectJ', *ACM SIGAPP Applied Computing Review*, vol. 9, no. 2, pp. 17-23.

Ramachandran, R., Pearce, D.J. & Welch, I. (2006) 'AspectJ for Multilevel Security', The 5th AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS), Bonn, Germany, 2006, 21 March 2006.

Rjaibi, W. & Bird, P. (2004) 'A Multi-Purpose Implementation of Mandatory Access Control in Relational Database Management Systems', in *Proceedings of 30th VLDBases Conference*, Toronto, Canada, pp. 1010-20.

Robinson, P., Rits, M. & Kilian-Kehr, R. (2004) 'An Aspect of Application Security Management', Proceedings of the 2nd International Workshop AOSDSEC'04, Lancaster, UK, March 2004.

Russell, D.F. & Gangemi, G.T. (1991) *Computer Security Basics*, O'Reilly Media and Associate, Sebastopol, California.

Samarati, P., Bertino, E., Ciampichetti, A. & Jajodia, S. (1997) 'Information Flow Control in Object-Oriented Systems', *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, no. 4, pp. 524-38.

Samarati, P. & de Capitani di Vimercati, S. (2001) 'Access control: Policies, models, and mechanisms', in Focardi, R. & Gorrieri, R. (eds), *Foundations of Security Analysis and Design*, Springer-Verlag, Berlin, vol. 2172, Lecture Notes in Computer Science, pp. 137-96.

Sandhu, R. & Park, J. (2003) 'Usage Control: A Vision for Next Generation Access Control', in Gorodetsky, V., Popyack, L.J. & Skormin, V.A. (eds), *Computer Network Security*, Springer, Berlin/Heidelberg, vol. 2776, Lecture notes in Computer Science, pp. 17-31.

Sandhu, R.S. (2001) 'Future Directions in Role-Based Access Control Models', in Gorodetski, V.I., Skormin, V.A. & Popyack, L.J. (eds), *Information Assurance in Computer Networks*, Springer, Berlin, Heidelberg, vol. 2052, Lecture Notes in Computer Science, pp. 22-6.

Sandhu, R.S., Coyne, E.J., Feinstein, H.L. & Youman, C.E. (1996) 'Role-Based Access Control Models', *IEEE computer*, vol. 29, no. 2, pp. 38-47.

Shah, V. & Hill, F. (2003) *An Aspect-Oriented Security Assurance Solution, Defence Advanced Research Projects Agency*, viewed 1 December 2005, <[Online] Available: <http://www.stormingmedia.us/50/5039/A503914.html>>.

Shin, W. & Yoo, S.B. (2007) 'Secured Web Services Based on Extended Usage Control', in Washio, T.Z., Z-H., Huang, J.Z., Hu, X., Li, J., Xie, C., He, J., Zou, D., Li, K.-C. & Freire, M.M. (eds), *Emerging Technologies in Knowledge Discovery and Data Mining, PAKDD 2007, International Workshops, Nanjing, China, May 22-25, 2007*, Springer, Berlin, vol. 4819, Lecture Notes in Computer Science, pp. 656-63.

Singh, A. (2005) 'The Scalability of AspectJ', MSC thesis, University of California, Davis.

Slowikowski, P. & Zielinski, K. (2003) 'Comparison Study of Aspect-oriented and Container Managed Security', Proceedings of the ECOOP workshop on analysis of Aspect-Oriented Software, Darmstadt, Germany, 21-25 July 2003.

Stevens, G. & Wulf, V. (2002) 'A New Dimension in Access Control: Studying Maintenance Engineering across Organizational Boundaries', Proceedings of the ACM conference on Computer Supported Cooperative Work (CSCW), New Orleans, Louisiana, USA, 16 -20 November 2002.

Syalim, A., Tabata, T. & Sakurai, K. (2005) 'Usage Control Model and Architecture for Data Confidentiality in a Database Service Provider', in *Indonesia Cryptology and Information Security Conference*, Jakarta, Indonesia, 30-31 March 2005, pp. 155-60.

Tolone, W., Ahn, G.-J., Pai, T. & Hong, S.-P. (2005) 'Access Control in Collaborative Systems', *Acm Computing Surveys*, vol. 37, no. 1, pp. 29-41.

Tymann, P.T. & Schneider, G.M. (2008) *Modern Software Development using Java*, 2nd edn, Thomson Course Technology, Boston, Massachusetts.

Ubayashi, N., Masuhara, H. & Tamai, T. (2004) 'An AOP Implementation Framework for Extending Joint Point Models', Proceedings of the ECOOP' 2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution, Oslo, Norway, 15 June 2004.

Vanhaute, B. & De Win, B. (2001) 'AOP, Security and Genericity', 1st Belgian AOSD Workshop, Vrije Universiteit Brussel, Brussels, Belgium, 8 November 2001.

Verhanneman, T., Piessens, F., De Win, B. & Joosen, W. (2005) 'Uniform Application-level Access Control Enforcement of Organizationwide Policies', in *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC 2005)*, Tucson, Arizona, 5-9 December 2005, pp. 431-40.

Viega, J., Bloch, J.T. & Chandra, P. (2001) 'Applying Aspect-Oriented Programming to Security', *Cutter IT Journal*, vol. 14, no. 2, pp. 31-9.

Viega, J. & Evans, D. (2000) 'Separation of concerns for security', in Tarr, P., Harrison, W., Ossher, H., Finkelstein, A., Nuseibeh, B. & Perry, D. (eds), *ICSE 2000 Workshop on Multi-Dimensional Separation of Concerns in Software Engineering*, Limerick, Ireland, 10 June 2000, pp. 125-38.

Viega, J. & Voas, J. (2000) 'Can Aspect-Oriented Programming Lead to More Reliable Software', *IEEE Software*, vol. 17, no. 6, pp. 19-21.

Wakaba, N. 2004, 'A Taxonomy of Aspect-Oriented Security (Honours Project), Unpublished Dissertation', University of South Africa.

Walker, R.J., Baniassad, E.L.A. & Murphy, G.C. (1999) 'An initial assessment of aspect-oriented programming', in *Proceedings of the 21st international conference on Software engineering*, Los Angeles, California, 16-22 May 1999, pp. 120-30.

Wang, H., Zhang, Y. & Cao, J. (2006) 'Ubiquitous Computing Environments and Its Usage Access Control', in *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, Hong Kong, 29 May - 1 June 2006, pp. 6-16.

Wegner, P. (1990) 'Concepts and Paradigms of Object-Oriented Programming', *ACM SIGPLAN OOPS Messenger*, vol. 1, no. 1, pp. 7-87.

Weippl, E. & Essmayr, W. (2003) 'Personal Trusted Devices for Web Services: Revisiting Multilevel Security', *Mobile Networks and Applications*, vol. 8, no. 2, pp. 151-7.

Whitten, A. & Tygar, J.D. (1999) 'Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0', Proceedings of the 8th USENIX Security Symposium, Washington DC, America, 23-26 August 1999.

Xu, Z., Feng, D., Li, L. & Chen, H. (2003) 'UC-RBAC: A Usage Constrained Role-Based Access Control Model', in Qing, S., Gollmann, D. & Zhou, J. (eds), *Information and Communications Security*, Springer, Berlin, vol. 2836, Lecture Notes in Computer Science, pp. 337-47.

Zakrzewski & Haddad, I. (2002) *Linux Distributed Security Module*, viewed 26 October 2007, <[Online]. Available: <http://www.linuxjournal.com/article/6215>>.

Zhang, X., Nakae, M., Covington, M.J. & Sandhu, R. (2006) 'A Usage-based Authorization Framework for Collaborative Computing Systems', in *Proceeding of Symposium on Access Control Models and Technologies (SACMAT'06)*, Lake Tahoe, California, USA, 7-9 June 2006, pp. 180-9.

Zhao, B., Sandhu, R., Zhang, X. & Qin, X. (2007) 'Towards a Times-Based Usage Control Model', in Barker, S. & Ahn, G.-J. (eds), *Data and Applications Security XXI*, Springer, Berlin/Heidelberg, vol. 4602, Lecture Notes in Computer Science, pp. 227-42.

Zhao, X. & Johnson, M.E. (2008) 'Access Flexibility with Escalation and Audit', in *20th Workshop on Information Systems and Economics (WISE 2008)*, Paris, France, 13-14 December 2008.



Zheng, L.M. & Myers, A.C. (2004) *Dynamic Security Labels and Noninterference*, Technical Report 2004., viewed 26 October 2005, <[Online]. Available: www.cs.cornell.edu/andru/papers/dynlabel.pdf>.

Zurko, M.E. (2005) 'User-Centered Security: Stepping Up to the Grand Challenge', 21st Annual Computer Security Applications Conference 2005, Tucson, Arizona, USA, 5-9 December 2005.

INDEX

A

access control · iii, iv, 2, 3, 4, 5, 6, 7, 11, 15, 17, 18, 19, 23, 30, 49, 50, 51, 52, 53, 67, 68, 73, 76
Discretionary access control · 5
mandatory access control · 18
Mandatory access control · 5
Optimistic access control · 8, 23
Access control · 9, 15
advice · 44, 45
Advice · 43, 45
Aspect · 4, 10, 12, 40, 41, 42, 43, 44, 46, 50, 51, 52, 53, 74, 75, 77, 79, 83, 85, 86
aspect orientation · 69, 70, 79
AspectJ · 43, 44, 45
aspect-orientation · iv, 9, 41, 48, 50, 54
aspect-oriented programming · 4, 5, 9, 12, 38, 40, 41, 46, 47, 48, 49, 52, 53, 54, 55, 68, 70, 76, 79, 93, 96, 99
Aspect-oriented programming · 40, 43, 50, 53
authentication · 4, 49, 50, 51

B

Break-the-Glass · 62, 63, 65, 73, 76

C

corrective control · 68

D

detective control · 68
deterrent control · 8, 14, 23, 68

I

information flow controls · 50

J

join points · 53
Join Points · 43

M

Mandatory access control · 15
mixed initiative access control framework · iii, 3, 14
mixed-initiative access control framework · 95, 97, 98

O

OAC(UCON) model · 9, 12, 26, 38, 60, 63, 71, 95, 96, 98
object-orientated paradigm · 40
object-oriented paradigm · 41, 48
ongoing conditions · 58, 63, 65, 72, 76, 78, 82
ongoing obligations · 67, 73
ongoing Obligations · 65
optimistic access control · iii, iv, 3, 5, 8, 9, 10, 11, 14, 21, 23, 25, 27, 28, 29, 37, 57, 62, 63, 64, 69, 77, 95, 96, 97
optimistic rights · 60, 63, 97, 98

P

Pointcut · 43
pointcuts · 44, 46
post-obligations · 66
pre-conditions · 24, 58, 63, 65, 72, 81
pre-obligations · 24, 65, 67, 72, 73, 92
Pre-Obligations · 64
preventative control · 68

R

role-based access control · 6, 15, 19, 20, 22, 36, 52, 63

S

structured programming · 41
Structured programming · 41

T

trust · iii, 2, 3, 6, 7, 8, 10, 23, 31



U

usage control · iii, iv, 2, 3, 5, 7, 8, 9, 10, 11, 14, 23, 24,
26, 28, 29, 30, 32, 34, 36, 37, 38, 58, 63, 64, 68, 69, 85,
92, 93, 95, 96, 97, 98, 99

W

Weaver · 44
Weaving · 44

APPENDIX A: PUBLICATIONS

The next challenge: Aspect-Oriented Programming

Abstract: Computer Science educationists face many challenges due to the rapid evolution in technology. One of the more recent challenges was the introduction of object-oriented programming to the computing curriculum. There have been many articles based on the difficulties encountered in teaching object-oriented programming and many solutions proposed in response. While some problems remain unresolved, the pressure to keep abreast of technology remains. The next hurdle that academics may face will be incorporating aspect-oriented programming into the curriculum. Although aspect-oriented programming is not yet ubiquitous in industry it is receiving considerable attention from research and practitioner communities alike. Increasingly academics will encounter the tension between teaching the fundamentals and introducing real-world technologies such as aspect-oriented programming that address real-world concerns. This paper addresses this particular notion, together with the challenges that will be faced if aspect-oriented programming is introduced into the computer science curriculum.

Reference:

Padayachee K. & Eloff J.H.P. 2006. The Next Challenge: Aspect-Oriented Programming, In: The Sixth IASTED International Conference on MODELLING, SIMULATION, AND OPTIMIZATION (MSO 2006) ACTA Press, Gaborone, Botswana, 11-13 September 2006, pages 123-127

An Aspect-Oriented Implementation of e-Consent to Foster Trust

Abstract: As society becomes increasingly dependent on software, there is an increasing expectation of information systems to protect the individual's right to privacy. The process of attaining electronic consent (e-Consent) may perhaps improve the trust that society has in information systems to protect these rights. However, an issue such as e-Consent is usually not given due consideration, as it is a non-functional issue and the implementation of the e-consent mechanism in disparate and legacy systems is difficult. Hence many systems are implemented without such types of controls. Evidently, aspect-oriented software design is highly extensible, as security concerns may be easily integrated into a completed software product. In this paper it is proposed that aspect-oriented programming be used to augment an existing system with electronic consent.

Reference: Padayachee, K. & Eloff J.H.P. 2006. Aspect-Oriented Implementation of e-Consent to foster Trust, In: SAICSIT 2006: Service-oriented software and Systems, Cape Town, South Africa, 9 - 11 October 2006, pages 164-169

An Aspect-Oriented Model to Monitor Misuse

Abstract: The efficacy of the aspect-oriented paradigm has been well established within several areas of software security as aspect-orientation facilitates the abstraction of these security-related tasks to reduce code complexity. The aim of this paper is to demonstrate that aspect-orientation may be used to monitor the information flows between objects in a system for the purposes of misuse detection. Misuse detection involves identifying behavior that is close to some previously defined pattern signature of a known intrusion.

Reference: Padayachee, K. & Eloff, J.H.P. 2006. An Aspect-Oriented Model to Monitor Misuse, International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering, In: Innovations and Advanced Techniques in Computer and Information Sciences and Engineering, Springer (Netherlands), pages: 273 -278, December 2006

An Aspect-Oriented Approach to Enhancing Multilevel Security with Usage Control: An Experience Report

Abstract: The aim of this paper is to document experiences with augmenting multilevel security with usage control at the application level within the aspect-oriented paradigm. Multilevel access control is an access control policy that supports systems that process especially sensitive data. However, attribute-based access control is sometimes insufficient and needs to be combined with additional features in order to meet the demands of modern applications and systems. Usage control enables finer-grained control over the usage of digital objects than do traditional access control policies and models.

Reference:

Padayachee, K. & Eloff, J.H.P. 2007. An Aspect-Oriented Approach to Enhancing Multilevel Security with Usage Control: An Experience Report, In: IAGENG: Lecture Notes in Engineering and Computer Science Volume 1 - International, Conference on Software Engineering (ICSE'07), Hong Kong, 21 - 23 March 2007, Hong Kong: Newswood Ltd. International Association of Engineers (Hong Kong), pages: 1060 - 1065

Enhancing Optimistic Access Controls with Usage Control

Abstract: With the advent of agile programming, lightweight software processes are being favoured over the highly formalised approaches of the past. Likewise, access control may benefit from a less prescriptive approach with an increasing reliance on users to behave ethically. These ideals correlate with optimistic access controls. However, ensuring that users behave in a trustworthy manner may require more than optimistic access controls. This paper investigates the possibility of enhancing optimistic access controls with usage control to ensure that users conduct themselves in a trustworthy manner. Usage control enables finer-grained control over the usage of digital objects than do traditional access control policies and models. Further to ease the development and maintenance of usage control measures, it is posited that it is completely separated from the application logic by using aspect-oriented programming.

Reference: Padayachee, K. and Eloff J.H.P. 2007. Enhancing Optimistic Access Controls with Usage Control, In: Lecture Notes in Computer Science: Trust, Privacy and Security in Digital Business, Springer (Germany), Volume 4657 Regensburg, Germany, September 3-7, 2007, pages: 75 – 82.

Adapting Usage Control as a Deterrent to address the Inadequacies of Access Controls

Abstract: Access controls are difficult to implement and evidently deficient under certain conditions. Traditional controls offer no protection for unclassified information, such as a telephone list of employees that is unrestricted, yet available only to members of the company. On the opposing side of the continuum, organizations such as hospitals that manage highly sensitive information require stricter access control measures. Yet, traditional access control may well have inadvertent consequences in such a context. Often, in unpredictable circumstances, users that are denied access could have prevented a calamity had they been allowed access. It has been proposed that controls such as auditing and accountability policies be enforced to deter rather than prevent unauthorized usage. In dynamic environments preconfigured access control policies may change dramatically depending on the context. Moreover, the cost of implementing and maintaining complex preconfigured access control policies sometimes far outweighs the benefits. This paper considers an adaptation of usage control as a proactive means of deterrence control to protect information that cannot be adequately or reasonably protected by access control.

Reference: Padayachee K, Eloff JHP (2009), Adapting usage control as a deterrent to address the inadequacies of access controls, Computers and Security (2009), Vol 28, No. 7, pages 536-544

APPENDIX B:

OOP DOCUMENTATION

For full documentation refer to the accompanying CD.

Hierarchy For All Packages

Package Hierarchies:

[accessobject](#), [authenticationSim](#), [authorizationSim](#), [components](#), [testutilities](#), [usagecontrol](#)

Class Hierarchy

- class java.lang.Object
 - class accessobject.[AccessInformation](#) (implements java.lang.Runnable)
 - class accessobject.[Access](#)
 - class usagecontrol.[BreakTheGlass](#)
 - class usagecontrol.[OngoingConditions](#)
 - class usagecontrol.[OngoingObligations](#)
 - class javax.swing.plaf.basic.BasicComboBoxEditor (implements javax.swing.ComboBoxEditor, java.awt.event.FocusListener)
 - class components.[JSearchableComboBox.SearchEditor](#)
 - class components.[CharUtility](#)
 - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
 - class java.awt.Container
 - class javax.swing.JComponent (implements java.io.Serializable)
 - class javax.swing.JComboBox (implements javax.accessibility.Accessible, java.awt.event.ActionListener, java.awt.ItemSelectable, javax.swing.event.ListDataListener)
 - class components.[JSearchableComboBox](#)
 - class javax.swing.JPanel (implements javax.accessibility.Accessible)
 - class components.[CheckBox](#) (implements java.awt.event.ItemListener)
 - class components.[Demo](#)
 - class components.[Image](#) (implements java.awt.event.ActionListener)
 - class java.awt.Window (implements javax.accessibility.Accessible)
 - class java.awt.Frame (implements java.awt.MenuContainer)
 - class javax.swing.JFrame (implements javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants)
 - class components.[ImageFrame](#)
 - class components.[DoublyLinkedList](#)
 - class components.[DoublyLinkedList.DLLIterator](#)
 - class components.[DoublyLinkedList.DLLNode](#)
 - class testutilities.[MemoryUsage](#)
 - class authorizationSim.[MyCallbackHandler](#) (implements javax.security.auth.callback.CallbackHandler)
 - class authorizationSim.[SampleAuthorization](#) (implements java.security.PrivilegedAction)
 - class authorizationSim.[SampleAzn](#)
 - class authenticationSim.[SampleLoginModule](#) (implements javax.security.auth.spi.LoginModule)
 - class authorizationSim.[SamplePrincipal](#) (implements java.security.Principal, java.io.Serializable)
 - class components.[TernarySearchTree](#)
 - class components.[TernarySearchTree.TSTNode](#)



- class usagecontrol.[UsageControl](#)

Java Documentation for Class BreakTheGlass

usagecontrol

Class BreakTheGlass

java.lang.Object

└─ [accessobject.AccessInformation](#)

└─ **usagecontrol.BreakTheGlass**

All Implemented Interfaces:

java.lang.Runnable

public class **BreakTheGlass**
extends [AccessInformation](#)

Field Summary

Fields inherited from class [accessobject.AccessInformation](#)

[AccessType](#), [aThread](#), [ObjectName](#), [SubjectName](#)

Constructor Summary

| | |
|-------------------|--|
| (package private) | BreakTheGlass (java.lang.String SubjectName, java.lang.String ObjectName, java.lang.String AccessType) |
|-------------------|--|

Method Summary

| | |
|-------------------|----------------------------|
| (package private) | display () |
| boolean | |

Methods inherited from class [accessobject.AccessInformation](#)

[endrequest](#), [getAccessType](#), [getObject](#), [getSubjectName](#), [run](#)



Methods inherited from class `java.lang.Object`

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`,
`wait`, `wait`, `wait`

Constructor Detail

BreakTheGlass

BreakTheGlass (`java.lang.String` SubjectName,
`java.lang.String` ObjectName,
`java.lang.String` AccessType)

Method Detail

display

`boolean display()`

Java Documentation for Class `OngoingConditions`

usagecontrol

Class `OngoingConditions`

`java.lang.Object`

└ [accessobject.AccessInformation](#)

└ `usagecontrol.OngoingConditions`

All Implemented Interfaces:

`java.lang.Runnable`

public class `OngoingConditions`

extends [AccessInformation](#)

Author:

Keshnee Padayachee

Field Summary

| | |
|--|--|
| <code>private</code> <code>static long</code> | condition Controls actions relating to the conditions of access |
| <code>private</code> <code>boolean</code> | stop |

Fields inherited from class `accessobject.AccessInformation`

[AccessType](#), [aThread](#), [ObjectName](#), [SubjectName](#)



Constructor Summary

[OngoingConditions](#) (java.lang.String SubjectName,
java.lang.String ObjectName, java.lang.String AccessType)

Method Summary

| | |
|---------|--|
| boolean | conditionIsValid() |
| void | conditionsWarning() |
| void | endOngoingConditions() |
| long | getCondition() |
| void | run() |

Methods inherited from class `accessobject.AccessInformation`

[endrequest](#), [getAccessType](#), [getObject](#), [getSubjectName](#)

Methods inherited from class `java.lang.Object`

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Field Detail

condition

private static long **condition**
Controls actions relating to the conditions of access

stop

private volatile boolean **stop**

Constructor Detail

OngoingConditions

```
public OngoingConditions (java.lang.String SubjectName,
                          java.lang.String ObjectName,
                          java.lang.String AccessType)
```



Method Detail

conditionsWarning

public void **conditionsWarning**()

getCondition

public long **getCondition**()

conditionIsValid

public boolean **conditionIsValid**()

run

public void **run**()

Specified by:

run in interface java.lang.Runnable

Overrides:

[run](#) in class [AccessInformation](#)

endOngoingConditions

public void **endOngoingConditions**()

Java Documentation for Class OngoingObligations

usagecontrol

Class OngoingObligations

java.lang.Object

└ [accessobject.AccessInformation](#)

└ **usagecontrol.OngoingObligations**

All Implemented Interfaces:

java.lang.Runnable

public class **OngoingObligations**

extends [AccessInformation](#)

Field Summary

| | | |
|----------------------------|---|--|
| private | OngoingObligationsRequest | Controls actions relating the OngoingObligations of the Access |
| ImageFrame | | |

Fields inherited from class [accessobject.AccessInformation](#)

[AccessType](#), [aThread](#), [ObjectName](#), [SubjectName](#)

Constructor Summary

[OngoingObligations](#)(java.lang.String SubjectName,
java.lang.String ObjectName,java.lang.String AccessType)



Method Summary

| | |
|------|--|
| void | endOngoingObligations () |
| void | run () |

Methods inherited from class `accessobject.AccessInformation`

[endrequest](#), [getAccessType](#), [getObject](#), [getSubjectName](#)

Methods inherited from class `java.lang.Object`

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Field Detail

OngoingObligationsRequest

private [ImageFrame](#) **OngoingObligationsRequest**
Controls actions relating the OngoingObligations of the Access

Constructor Detail

OngoingObligations

```
public OngoingObligations (java.lang.String SubjectName,  
                           java.lang.String ObjectName,  
                           java.lang.String AccessType)
```

Method Detail

run

```
public void run ()  
    Specified by:  
    run in interface java.lang.Runnable  
    Overrides:  
    run in class AccessInformation
```

endOngoingObligations

```
public void endOngoingObligations ()
```

Java Documentation Class UsageControl

usagecontrol

Class UsageControl

java.lang.Object

└─ usagecontrol.UsageControl

public class **UsageControl**
extends java.lang.Object

Field Summary

| | |
|---|--|
| private Access | accessObject |
| private static boolean | accessOpen Controls the Usage control of an Access to an Object |
| private java.lang.Thread | accessThread |
| (package private) java.lang.String | AccessType |
| private BreakTheGlass | breakTheGlass |
| static boolean | conditionsInvalid |
| private java.lang.Thread | conditionsThread |
| static boolean | endAccess |
| static boolean | endObligations |
| (package private) java.lang.String | ObjectName |
| private java.lang.Thread | obligationsThread |
| private OngoingConditions | OnConditions |
| private OngoingObligations | Onobligations |
| private static boolean | preCondition |
| (package private) java.lang.String | SubjectName |

Constructor Summary

[UsageControl](#)([Access](#) AccessObject)



| | |
|--|--|
| | |
|--|--|

| Method Summary | |
|------------------------------|---|
| (package private) boolean | breakTheGlass (java.lang.String SubjectName, java.lang.String ObjectName, java.lang.String AccessType) |
| boolean | checkAccessType () |
| (package private) void | initiateBreakTheGlassFacility () |
| boolean | initiateUsageControl () |
| (package private) void | logAccess (java.lang.String SubjectName, java.lang.String ObjectName, java.lang.String AccessType, java.lang.String Notice, java.lang.String RedFlag) |
| (package private) void | postAccess () |
| void | postObligations (java.lang.String SubjectName, java.lang.String ObjectName, java.lang.String AccessType) |
| (package private) boolean | preConditions (java.lang.String SubjectName, java.lang.String ObjectName, java.lang.String AccessType) |
| (package private) boolean | preObligations (java.lang.String SubjectName, java.lang.String ObjectName, java.lang.String AccessType) |
| (package private) void | stopAccess () |
| (package private) void | stopOngoingObligations () |

| Methods inherited from class java.lang.Object |
|--|
| clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

| Field Detail |
|-------------------|
| accessOpen |



```
private static boolean accessOpen  
    Controls the Usage control of an Access to an Object
```

preCondition

```
private static boolean preCondition
```

obligationsThread

```
private java.lang.Thread obligationsThread
```

conditionsThread

```
private java.lang.Thread conditionsThread
```

accessThread

```
private java.lang.Thread accessThread
```

OnConditions

```
private OngoingConditions OnConditions
```

accessObject

```
private Access accessObject
```

Onobligations

```
private OngoingObligations Onobligations
```

breakTheGlass

```
private BreakTheGlass breakTheGlass
```

SubjectName

```
java.lang.String SubjectName
```

ObjectName

```
java.lang.String ObjectName
```

AccessType

```
java.lang.String AccessType
```

conditionsInvalid

```
public static boolean conditionsInvalid
```

endAccess

```
public static boolean endAccess
```

endObligations

```
public static boolean endObligations
```

Constructor Detail

UsageControl

```
public UsageControl(Access AccessObject)
```

Method Detail

checkAccessType

```
public boolean checkAccessType()
```

initiateUsageControl

```
public boolean initiateUsageControl()
```

initiateBreakTheGlassFacility

```
void initiateBreakTheGlassFacility()
```

stopAccess

```
void stopAccess()
```

stopOngoingObligations

```
void stopOngoingObligations()
```

preObligations

```
boolean preObligations(java.lang.String SubjectName,  
                        java.lang.String ObjectName,  
                        java.lang.String AccessType)
```

preConditions

```
boolean preConditions(java.lang.String SubjectName,  
                     java.lang.String ObjectName,  
                     java.lang.String AccessType)
```

postAccess

```
void postAccess()
```

postObligations

```
public void postObligations(java.lang.String SubjectName,  
                            java.lang.String ObjectName,  
                            java.lang.String AccessType)
```

breakTheGlass

```
boolean breakTheGlass(java.lang.String SubjectName,  
                      java.lang.String ObjectName,  
                      java.lang.String AccessType)
```

logAccess

```
void logAccess(java.lang.String SubjectName,  
              java.lang.String ObjectName,  
              java.lang.String AccessType,  
              java.lang.String Notice,  
              java.lang.String RedFlag)
```

Source Code for class Access

```
package accessobject;  
import javax.swing.UIManager;  
import usagecontrol.UsageControl;  
import components.image;  
  
public class Access extends AccessInformation {  
    /**  
     * This class controls the object being accessed  
     */  
    public void request(){  
        UIManager.put("swing.boldMetal", Boolean.FALSE);  
        image.createAndShowGUI(ObjectName);  
    }  
}
```

```
public Access(String SubjectName, String ObjectName, String AccessType) {
    super(SubjectName,ObjectName,AccessType) ;
}

public void endrequest() {
    super.endrequest();
    image.close();
}

public void run() {

    super.run();
    UIManager.put("swing.boldMetal", Boolean.FALSE);
    image.createAndShowGUI(ObjectName);

    aThread = Thread.currentThread();
    // Keep going as long as myThread is the same as the current thread.

    while (image.WindowOpen) {
        try {
            Thread.sleep(500); // Tell the thread to sleep for half a second.
        }
        catch (InterruptedException e) {}
    }

    if (!image.WindowOpen){
        endrequest();
        //object-oriented version
        UsageControl.endAccess = true;
        //end object-oriented version
    }
}
}
```

Source code for class AccessInformation

```
package accessobject;

public class AccessInformation implements Runnable{
    /**
     * This class maintains all the details relating to the access
     */
    protected String SubjectName;
    protected String ObjectName;
    protected String AccessType;
    protected Thread aThread;
    public AccessInformation(String subName, String OName, String type) {
        SubjectName = subName;
        ObjectName = OName;
        AccessType = type;
    }
    public String getSubjectName()
    {
        return SubjectName;
    }

    public String getObject()
    {
        return ObjectName;
    }
    public String getAccessType()
    {
        return AccessType;
    }
    public void run(){aThread = Thread.currentThread(); }

    public void endrequest() {
        aThread = null;
    }
}
```

Source code for class BreakTheGlass

```
package usagecontrol;
import javax.swing.ImageIcon;
import javax.swing.JOptionPane;

import accessobject.AccessInformation;

public class BreakTheGlass extends AccessInformation{
    /**
     * Provides the BreakTheGlass Interface
     */
    BreakTheGlass(String SubjectName, String ObjectName, String AccessType ) {
        super(SubjectName,ObjectName,AccessType);
    }

    boolean display(){
        String message = "<html>" + SubjectName + " are you <font color = green> SURE <font
        color=black>" +
        "you want to continue with this access?"
        + "<br>(a) This access will be <font color=red>RED-FLAGGED<font color=black>!!!"
        + "<br>(b) You will have justify this usage to the system adminstrator" ;

        ImageIcon icon = new ImageIcon("c:\\icons\\policestop.gif");

        int answer = JOptionPane.showConfirmDialog(null, message,"BREAK THE GLASS IN CASE OF
        EMERGENCY",
        JOptionPane.YES_NO_OPTION,JOptionPane.INFORMATION_MESSAGE, icon);

        if (answer == JOptionPane.YES_OPTION) {
            return true;
        }
        else if (answer == JOptionPane.NO_OPTION) {
            return false;
        }
        return false;
    }
}
```

Source code for class OngoingConditions

```
package usagecontrol;
import javax.swing.*;
import accessobject.AccessInformation;

public class OngoingConditions extends AccessInformation{
    /**
     * Controls actions relating to the conditions of access
     */
    private static long condition = 0;
    private volatile boolean stop = false;
    public OngoingConditions(String SubjectName, String ObjectName, String AccessType ) {
        super(SubjectName,ObjectName,AccessType);
    }

    // This will terminate the run() method.
    public void conditionsWarning(){
        ImageIcon icon = new ImageIcon("c:\\icons\\warn1.gif");
        String message = "<html> <font color=blue> "+ SubjectName
            +", is <font color = red> PROHIBITED<font color=blue> "
            + "from accessing client file: " + ObjectName + " after working hours";
        JOptionPane.showMessageDialog(null, message ,"CONDITIONS WARNING",
        JOptionPane.INFORMATION_MESSAGE,icon);

        //object-oriented version
        UsageControl.conditionsInvalid = true;
        //end of object-oriented version
    }

    public long getCondition(){
        condition++;
        return condition;
    }
}
```

```
public boolean conditionisValid()
{
    condition++;
    if (condition%10 == 0)
        return false;
    else
        return true;
}
public void run() {
    super.run();
    while(conditionisValid()){
        try {
            Thread.sleep(1000); // Tell the thread to sleep for a second.
        }
        catch (InterruptedException e) {}
    }
    if (!stop){
        conditionsWarning();
    }
}
public void endOngoingConditions() {
    stop = true;
}
}
```

Source code for class OngoingObligations

```
package usagecontrol;

import java.awt.Color;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class OngoingObligations extends AccessInformation{
    /**
     * Controls actions relating the OngoingObligations of the Access
     */
    private ImageFrame OngoingObligationsRequest;
    public OngoingObligations(String SubjectName, String ObjectName, String AccessType) {
        super(SubjectName, ObjectName, AccessType);
    }

    public void run() {
        super.run();
        String Message = "<html><font color = green>" + SubjectName + "   ACCESSING...client
file: "
            + ObjectName + " WITH RIGHTS "+           AccessType+". <br> ";
        Message.toUpperCase();
        ImageIcon icon = new ImageIcon("c:\\files\\OBS.jpg");
        OngoingObligationsRequest = new ImageFrame(600,300,400,400,"Ongoing
Obligations",Message,icon);
        OngoingObligationsRequest.setForeground(Color.BLUE);
        OngoingObligationsRequest.setResizable(false);
        // Keep going as long as myThread is the same as the current thread.

        while (OngoingObligationsRequest.windowOpen()) {
            try {
                Thread.sleep(500); // Tell the thread to sleep for half a second.
            }
            catch (InterruptedException e) {}
        }
        if (!OngoingObligationsRequest.windowOpen()){
            OngoingObligationsRequest.close();
            //object-oriented version
            UsageControl.endObligations = true;
            //end of object-oriented version
        }
    }
    public void endOngoingObligations()
    { UsageControl.endObligations = true;
      if (OngoingObligationsRequest.windowOpen()){
```

```

        OngoingObligationsRequest.close();
    }
}
}

```

Source Code for class UsageControl

```

package usagecontrol;

import javax.swing.JOptionPane;
import javax.swing.ImageIcon;
import accessobject.Access;
import components.CheckBox;

public class UsageControl {
    /**
     * Controls the Usage control of an Access to an Object
     */
    private static boolean accessOpen;
    private static boolean preCondition = true;
    private Thread obligationsThread;
    private Thread conditionsThread;
    private Thread accessThread;
    private OngoingConditions OnConditions;
    private Access accessObject;
    private OngoingObligations Onobligations;
    private BreakTheGlass breakTheGlass;
    String SubjectName;
    String ObjectName;
    String AccessType;

    public static boolean conditionsInvalid = false;
    public static boolean endAccess = false;
    public static boolean endObligations = false;

    public UsageControl(Access AccessObject)
    {
        accessObject = AccessObject;
        SubjectName = AccessObject.getSubjectName();
        ObjectName = AccessObject.getObject();
        AccessType = AccessObject.getAccessType();
    }

    public boolean checkAccessType(){
        return true;
    }

    public boolean initiateUsageControl(){
        accessOpen = true;
        conditionsInvalid = false;
        endObligations = false;
        endAccess = false;
        if (preObligations(SubjectName, ObjectName, AccessType))
        {
            if (preConditions(SubjectName, ObjectName, AccessType)
            || breakTheGlass(SubjectName, ObjectName, AccessType)){
                accessThread = new Thread(accessObject);
                accessThread.start();

                Onobligations = new OngoingObligations( SubjectName, ObjectName, AccessType);
                obligationsThread = new Thread(Onobligations);
                obligationsThread.start();

                OnConditions = new OngoingConditions(SubjectName,ObjectName,AccessType);
                conditionsThread = new Thread(OnConditions);
                conditionsThread.start();

                while(accessOpen){
                    try {
                        Thread.sleep(500);
                        if (conditionsInvalid){
                            initiateBreakTheGlassFacility();
                        }
                    }
                    if (endAccess){

```

```
        stopAccess();
    }
    if (endObligations){
        stopOngoingObligations();
    }

} catch (InterruptedException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
}
postObligations (SubjectName,ObjectName,AccessType);
}
else
    System.out.println("no conditions");
}
else
    System.out.println("no obligations");
return true;
}
}
/**
 *
 */

void initiateBreakTheGlassFacility(){

    if (!breakTheGlass (OnConditions.getSubjectName(), OnConditions.getObject(),
OnConditions.getAccessType()))
    {    Onobligations.endOngoingObligations();
        postAccess();
    }
    conditionsInvalid = false;
}

void stopAccess(){
    Onobligations.endOngoingObligations();
    if (accessOpen)
        postAccess();
}

void stopOngoingObligations(){
    if (accessOpen)
        postAccess();
}

boolean preObligations(String SubjectName, String ObjectName, String AccessType){

    ImageIcon icon = new ImageIcon("c:\\icons\\hand.gif");
    String message = "<html>" + SubjectName + ", if you click <font color=green> YES " +
        "<font color=black> you agree to <font                color=red>NOT<font
color=black>" +
        " distribute client file: "
        + ObjectName;
    int answer = JOptionPane.showConfirmDialog(null, message,"PRE-OBLIGATIONS",
        JOptionPane.YES_NO_OPTION,JOptionPane.WARNING_MESSAGE,icon);
    if (answer == JOptionPane.YES_OPTION) {
        return true;
    }
    else if (answer == JOptionPane.NO_OPTION) {
        return false;
    }
    return false;
}

boolean preConditions(String SubjectName, String ObjectName, String AccessType){

    if (preCondition) {
        ImageIcon icon = new ImageIcon("c:\\icons\\warn1.gif");
        String message = "<html> <font color=blue>" + SubjectName +", "
        +" <font color=red>PROHIBITED<font color = blue> from accessing client file: " +
        ObjectName + " at this time !" ;
        JOptionPane.showMessageDialog(null, message ,"PRE-CONDITIONS WARNING",
        JOptionPane.INFORMATION_MESSAGE,icon);
        preCondition = false;
    }
}
```



```
    return false;
}
return true;
}

void postAccess() {
    accessOpen = false;
    accessObject.endrequest();
    Onobligations.endOngoingObligations();
    OnConditions.endrequest();
    //update logs
}

public void postObligations(String SubjectName, String ObjectName, String AccessType) {
    CheckBox.createAndShowGUI();
}

boolean breakTheGlass(String SubjectName, String ObjectName, String AccessType) {
    breakTheGlass = new BreakTheGlass(SubjectName, ObjectName, AccessType);
    if (breakTheGlass.display()) {
        logAccess(SubjectName, ObjectName, AccessType, "Illegal Access", "YES");
        return true;
    }
    return false;
}

void logAccess(String SubjectName, String ObjectName, String AccessType, String Notice,
String RedFlag) {
    //WRITE TO LOG FILE
}
}
```

APPENDIX C:

AOP DOCUMENTATION

For full documentation refer to accompanying CD.

Hierarchy For All Packages

Package Hierarchies:

[accessobject](#), [authenticationSim](#), [authorizationSim](#), [components](#), [testutilities](#), [usagecontrol](#)

Class Hierarchy

- class java.lang.Object
 - class accessobject.[AccessInformation](#)
 - class accessobject.[Access](#)
 - class usagecontrol.[OngoingConditions](#) (implements java.lang.Runnable)
 - class usagecontrol.[BreakTheGlass](#)
 - class usagecontrol.[OngoingObligations](#) (implements java.lang.Runnable)
 - class javax.swing.plaf.basic.BasicComboBoxEditor (implements javax.swing.ComboBoxEditor, java.awt.event.FocusListener)
 - class components.[JSearchableComboBox.SearchEditor](#)
 - class components.[CharUtility](#)
 - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
 - class java.awt.Container
 - class javax.swing.JComponent (implements java.io.Serializable)
 - class javax.swing.JComboBox (implements javax.accessibility.Accessible, java.awt.event.ActionListener, java.awt.ItemSelectable, javax.swing.event.ListDataListener)
 - class components.[JSearchableComboBox](#)
 - class javax.swing.JPanel (implements javax.accessibility.Accessible)
 - class components.[CheckBox](#) (implements java.awt.event.ItemListener)
 - class components.[Demo](#)
 - class components.[Image](#) (implements java.awt.event.ActionListener)
 - class java.awt.Window (implements javax.accessibility.Accessible)
 - class java.awt.Frame (implements java.awt.MenuContainer)
 - class javax.swing.JFrame (implements javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants)
 - class components.[ImageFrame](#)
 - class components.[DoublyLinkedList](#)
 - class components.[DoublyLinkedList.DLLIterator](#)
 - class components.[DoublyLinkedList.DLLNode](#)
 - class usagecontrol.[IntertypeDeclarationOnAccess](#)
 - class testutilities.[MemoryUsage](#)
 - class authorizationSim.[MyCallbackHandler](#) (implements javax.security.auth.callback.CallbackHandler)
 - class authorizationSim.[SampleAuthorization](#) (implements java.security.PrivilegedAction)
 - class authorizationSim.[SampleAzn](#)
 - class authenticationSim.[SampleLoginModule](#) (implements javax.security.auth.spi.LoginModule)

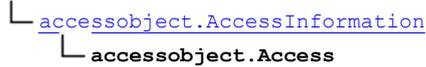


- class authorizationSim.[SamplePrincipal](#) (implements java.security.Principal, java.io.Serializable)
- class components.[TernarySearchTree](#)
- class components.[TernarySearchTree.TSTNode](#)
- class usagecontrol.[UsageControlInjector](#)

Java Documentation for Class Access:

Class Access

java.lang.Object



Direct Known Subclasses:

[OngoingConditions](#)

public class Access

extends [AccessInformation](#)

Advised by: [usagecontrol.UsageControlInjector.after\(\): OngoingAccess..](#)

Aspect declarations: [usagecontrol.IntertypeDeclarationOnAccess.declare](#) [parents: implements Runnable](#)

Author:

KESHNEE TODO To change the template for this generated type comment go to Window - Preferences - Java - Code Style - Code Templates

Inter-Type Method Summary

| | |
|------|--|
| void | Access.endrequest() Declared by: usagecontrol.IntertypeDeclarationOnAccess |
| void | Access.run() Declared by: usagecontrol.IntertypeDeclarationOnAccess |

Inter-Type Field Summary

| | |
|----------------|---|
| package Thread | Access.aThread Declared by: usagecontrol.IntertypeDeclarationOnAccess |
|----------------|---|

Field Summary

Fields inherited from class accessobject.[AccessInformation](#)

[accessType](#), [objectName](#), [subjectName](#)

Constructor Summary

[Access](#)(java.lang.String SubjectName, java.lang.String ObjectName, java.lang.String AccessType)



| Method Summary | |
|----------------|---|
| void | close() |
| void | request() |
| Advised by: | usagecontrol.UsageControlInjector.around(accessobject.Access) : Intercept Request.. |

| Methods inherited from class accessobject.AccessInformation |
|--|
| getAccessType , getObject , getSubjectName |

| Methods inherited from class java.lang.Object |
|---|
| <code>clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait</code> |

| Constructor Detail |
|---|
| Access public Access (java.lang.String SubjectName, java.lang.String ObjectName, java.lang.String AccesType) |

| Method Detail |
|--|
| request public void request () Advised by: usagecontrol.UsageControlInjector.around(accessobject.Access) : Intercept Request.. |

| |
|---|
| close public void close () |
|---|

Java Documentation for class [AccessInformation](#)

Class [AccessInformation](#)
 java.lang.Object
 └─ [accessobject.AccessInformation](#)

Direct Known Subclasses:
[Access](#), [BreakTheGlass](#), [OngoingObligations](#)

```
public class AccessInformation
  extends java.lang.Object
  Author:
    KESHNEE TODO To change the template for this generated type comment go to Window - Preferences - Java - Code Style - Code
    Templates
```

| Field Summary | |
|-------------------------------|----------------------------|
| protected java.lang.String | accessType |
| protected | objectName |



| | |
|----------------------------|-----------------------------|
| java.lang.String | |
| protected java.lang.String | subjectName |

| Constructor Summary | |
|-----------------------------------|---|
| AccessInformation | (java.lang.String subName, java.lang.String OName, java.lang.String type) |

| Method Summary | |
|------------------|-----------------------------------|
| java.lang.String | getAccessType () |
| java.lang.String | getObject () |
| java.lang.String | getSubjectName () |

| Methods inherited from class java.lang.Object |
|--|
| clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

| Field Detail |
|---|
| subjectName protected java.lang.String subjectName |
| objectName protected java.lang.String objectName |
| accessType protected java.lang.String accessType |

| Constructor Detail |
|---|
| AccessInformation public AccessInformation (java.lang.String subName, java.lang.String OName, java.lang.String type) |

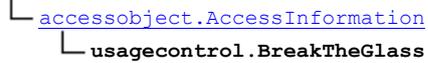
| Method Detail |
|---|
| getSubjectName public java.lang.String getSubjectName () |
| getObject public java.lang.String getObject () |
| getAccessType public java.lang.String getAccessType () |



Java Documentation for Class BreakTheGlass

Class BreakTheGlass

java.lang.Object



public class **BreakTheGlass**
extends [AccessInformation](#)

Field Summary

Fields inherited from class accessobject.[AccessInformation](#)

[accessType](#), [objectName](#), [subjectName](#)

Constructor Summary

| | |
|-------------------|--|
| (package private) | BreakTheGlass (java.lang.String SubjectName, java.lang.String ObjectName, java.lang.String AccessType) |
|-------------------|--|

Method Summary

| | |
|-------------------|---------------------------------------|
| (package private) | display () boolean |
|-------------------|---------------------------------------|

Methods inherited from class accessobject.[AccessInformation](#)

[getAccessType](#), [getObject](#), [getSubjectName](#)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

BreakTheGlass

BreakTheGlass (java.lang.String SubjectName, java.lang.String ObjectName, java.lang.String AccessType)

Method Detail

display



boolean **display()**

AJDocumentation on Aspect IntertypeDeclarationOnAccess

java.lang.Object

└─ usagecontrol.IntertypeDeclarationOnAccess

public aspect **IntertypeDeclarationOnAccess**

extends java.lang.Object

Author:

KESHNEE

Declare Summary

| | |
|----------------|--|
| | declare parents: implements Runnable Declared on: accessobject.Access |
| package Thread | Access.aThread Declared on: accessobject.Access |
| void | Access.endrequest() Declared on: accessobject.Access |
| void | Access.run() Declared on: accessobject.Access |

Constructor Summary

[IntertypeDeclarationOnAccess\(\)](#)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Declare Detail

declare parents: implements Runnable

Declared on: [accessobject.Access](#)

Access.aThread

package Thread **Access.aThread**

Declared on: [accessobject.Access](#)

Access.endrequest()

public void **Access.endrequest()**



Declared on: [accessobject.Access](#)

Access.run()

```
public void Access.run()
```

Declared on: [accessobject.Access](#)

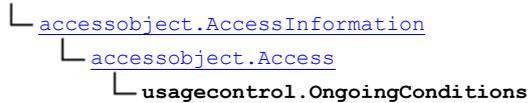
Constructor Detail

IntertypeDeclarationOnAccess

```
public IntertypeDeclarationOnAccess ()
```

Java Documentation on Class OngoingConditions

java.lang.Object



All Implemented Interfaces:

java.lang.Runnable

```
public class OngoingConditions
```

```
extends Access
```

```
implements java.lang.Runnable
```

Author:

PADAYK TODO To change the template for this generated type comment go to Window - Preferences - Java - Code Style - Code Templates

Field Summary

| | |
|---------------------|--|
| private static long | condition Controls actions relating to the conditions of access |
| private boolean | stop |

Fields inherited from class [accessobject.AccessInformation](#)

[accessType](#), [objectName](#), [subjectName](#)

Constructor Summary

```
OngoingConditions(java.lang.String SubjectName, java.lang.String ObjectName, java.lang.String AccessType)
```

Method Summary

| | |
|---------|--|
| boolean | conditionIsValid() |
| void | conditionsWarning() |
| void | endOngoingConditions() |



| | |
|------|--|
| | |
| long | getCondition() |
| void | run() |
| | Advised by: usagecontrol.UsageControlInjector.after(): OngoingCondition.. |

| |
|---|
| Methods inherited from class accessobject.Access |
| close , request |

| |
|--|
| Methods inherited from class accessobject.AccessInformation |
| getAccessType , getObject , getSubjectName |

| |
|--|
| Methods inherited from class java.lang.Object |
| clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

Field Detail

condition
private static long **condition**
Controls actions relating to the conditions of access

stop
private volatile boolean **stop**

Constructor Detail

OngoingConditions
public **OngoingConditions**(java.lang.String SubjectName,
java.lang.String ObjectName,
java.lang.String AccessType)

Method Detail

conditionsWarning
public void **conditionsWarning**()

getCondition
public long **getCondition**()

conditionisValid
public boolean **conditionisValid**()

run
public void **run**()
Advised by: [usagecontrol.UsageControlInjector.after\(\): OngoingCondition..](#)
Specified by:
run in interface java.lang.Runnable

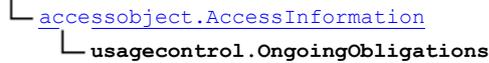
endOngoingConditions



```
public void endOngoingConditions()
```

Java Documentation on Class OngoingObligations

```
java.lang.Object
```



All Implemented Interfaces:

```
java.lang.Runnable
```

```
public class OngoingObligations
extends AccessInformation
implements java.lang.Runnable
```

Field Summary

| | |
|---------------------------------------|---|
| private ImageFrame | OngoingObligationsRequest Controls actions relating the OngoingObligations of the Access |
|---------------------------------------|---|

Fields inherited from class accessobject.[AccessInformation](#)

```
accessType, objectName, subjectName
```

Constructor Summary

```
OngoingObligations(java.lang.String SubjectName, java.lang.String ObjectName, java.lang.String AccessType)
```

Method Summary

| | |
|--------------------|--|
| void | endOngoingObligations () |
| void | run () |
| Advised by: | usagecontrol.UsageControlInjector.after () : OngoingObligation.. |

Methods inherited from class accessobject.[AccessInformation](#)

```
getAccessType, getObject, getSubjectName
```

Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
```



Field Detail

OngoingObligationsRequest
private [ImageFrame](#) **OngoingObligationsRequest**
Controls actions relating the OngoingObligations of the Access

Constructor Detail

OngoingObligations
public **OngoingObligations** (java.lang.String SubjectName,
java.lang.String ObjectName,
java.lang.String AccessType)

Method Detail

run
public void **run**()
Advised by: [usagecontrol.UsageControlInjector.after\(\): OngoingObligation..](#)
Specified by:
run in interface java.lang.Runnable

endOngoingObligations
public void **endOngoingObligations**()

AJDocumentation on Aspect UsageControllInjector

java.lang.Object
└─ [usagecontrol.UsageControlInjector](#)

public aspect **UsageControllInjector**
extends java.lang.Object

Pointcut Summary

| | |
|-------------------|---|
| (package private) | Intercept Request (accessobject.Access) |
| (package private) | OngoingCondition () |
| (package private) | OngoingAccess () |
| (package private) | OngoingObligation () |

Advice Summary

| | |
|---------------------------------------|---|
| around (accessobject.Access) : | Intercept Request.. |
| Advises: | accessobject.Access.request |
| after () : | OngoingCondition.. |
| Advises: | usagecontrol.OngoingConditions.run |
| after () : | OngoingAccess.. |
| Advises: | accessobject.Access |
| after () : | OngoingObligation.. |
| Advises: | usagecontrol.OngoingObligations.run |



| Field Summary | |
|---|-----------------------------------|
| private Access | accessObject |
| private static boolean | accessOpen |
| private java.lang.Thread | accessThread |
| private java.lang.String | AccessType |
| private java.lang.Thread | conditionsThread |
| private java.lang.String | ObjectName |
| private java.lang.Thread | obligationsThread |
| private OngoingConditions | OnConditions |
| private OngoingObligations | Onobligations |
| private static boolean | preCondition |
| private java.lang.String | SubjectName |

| Constructor Summary | |
|---|--|
| UsageControlInjector () | |

| Method Summary | |
|------------------------------|---|
| (package private) boolean | breakTheGlass (java.lang.String SubjectName, java.lang.String ObjectName, java.lang.String AccessType) |
| (package private) void | initiateBreakTheGlassFacility () |
| (package private) void | logAccess (java.lang.String SubjectName, java.lang.String ObjectName, java.lang.String AccessType, java.lang.String Notice, java.lang.String RedFlag) |
| boolean | OptimisticRights () |
| (package private) void | postAccess () |
| void | postObligations (java.lang.String SubjectName, java.lang.String ObjectName, java.lang.String AccessType) |



| | |
|---------------------------|---|
| (package private) boolean | preConditions (java.lang.String SubjectName, java.lang.String ObjectName, java.lang.String AccessType) |
| (package private) boolean | preObligations (java.lang.String SubjectName, java.lang.String ObjectName, java.lang.String AccessType) |

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Pointcut Detail

Intercept_Request(accessobject.Access)

OngoingCondition()

OngoingAccess()

OngoingObligation()

Advice Detail

around

around(accessobject.Access): Intercept_Request..

Advises: [accessobject.Access.request](#)

after

after(): OngoingCondition..

Advises: [usagecontrol.OngoingConditions.run](#)

after

after(): OngoingAccess..

Advises: [accessobject.Access](#)

after

after(): OngoingObligation..

Advises: [usagecontrol.OngoingObligations.run](#)



Field Detail

accessOpen

```
private static boolean accessOpen
```

preCondition

```
private static boolean preCondition
```

obligationsThread

```
private java.lang.Thread obligationsThread
```

conditionsThread

```
private java.lang.Thread conditionsThread
```

accessThread

```
private java.lang.Thread accessThread
```

accessObject

```
private Access accessObject
```

OnConditions

```
private OngoingConditions OnConditions
```

Onobligations

```
private OngoingObligations Onobligations
```

SubjectName

```
private java.lang.String SubjectName
```

ObjectName

```
private java.lang.String ObjectName
```

AccessType

```
private java.lang.String AccessType
```

Constructor Detail

UsageControlInjector

```
public UsageControlInjector()
```

Method Detail

OptimisticRights

```
public boolean OptimisticRights()
```

initiateBreakTheGlassFacility

```
void initiateBreakTheGlassFacility()
```

preObligations

```
boolean preObligations(java.lang.String SubjectName,  
                        java.lang.String ObjectName,  
                        java.lang.String AccessType)
```

preConditions

```
boolean preConditions(java.lang.String SubjectName,  
                      java.lang.String ObjectName,  
                      java.lang.String AccessType)
```

postAccess

```
void postAccess()
```

postObligations

```
public void postObligations(java.lang.String SubjectName,  
                             java.lang.String ObjectName,
```

```
java.lang.String AccessType)
```

breakTheGlass

```
boolean breakTheGlass(java.lang.String SubjectName,  
                        java.lang.String ObjectName,  
                        java.lang.String AccessType)
```

logAccess

```
void logAccess(java.lang.String SubjectName,  
              java.lang.String ObjectName,  
              java.lang.String AccessType,  
              java.lang.String Notice,  
              java.lang.String RedFlag)
```

Source Code for class Access

```
package accessobject;  
  
import javax.swing.UIManager;  
  
import components.image;  
  
public class Access extends AccessInformation{  
  
    public void request(){  
        System.out.println("Inside request");  
        UIManager.put("swing.boldMetal", Boolean.FALSE);  
        image.createAndShowGUI(objectName);  
    }  
  
    public Access(String SubjectName, String ObjectName, String AccessType) {  
        super(SubjectName, ObjectName, AccessType) ;  
    }  
  
    public void close(){  
        image.close();  
    }  
}
```

Source code class AccessInformation

```
package accessobject;  
  
public class AccessInformation {  
    protected String subjectName;  
    protected String objectName;  
    protected String accessType;  
    public AccessInformation(String subName, String OName, String type) {  
        subjectName = subName;  
        objectName = OName;  
        accessType = type;  
    }  
    public String getSubjectName() {  
        {  
            return subjectName;  
        }  
    }  
  
    public String getObject() {  
        {  
            return objectName;  
        }  
    }  
    public String getAccessType() {  
        {  
            return accessType;  
        }  
    }  
}
```

Source Code for BreakTheGlass

```
package usagecontrol;
import javax.swing.ImageIcon;
import javax.swing.JOptionPane;

import accessobject.AccessInformation;
public class BreakTheGlass extends AccessInformation{
    /**
     * Provides the BreakTheGlass Interface
     */
    BreakTheGlass(String SubjectName, String ObjectName, String AccessType ) {
        super(SubjectName,ObjectName,AccessType);
    }

    boolean display(){
    String message = "<html>" + subjectName + " are you <font color = green> SURE <font
    color=black>" +
        "you want to continue with this access?"
        +"<br>(a) This access will be <font color=red>RED-FLAGGED<font color=black>!!!"
        +"<br>(b) You will have to justify this usage to the system administrator" ;

        ImageIcon icon = new ImageIcon("c:\\test0\\policestop.gif");

        int answer = JOptionPane.showConfirmDialog(null, message,"BREAK THE GLASS IN CASE OF
        EMERGENCY",
        JOptionPane.YES_NO_OPTION,JOptionPane.INFORMATION_MESSAGE, icon);

        if (answer == JOptionPane.YES_OPTION) {
            return true;
        }
        else if (answer == JOptionPane.NO_OPTION) {
            return false;
        }
        return false;
    }
}
```

Source code for InterTypeDeclarationOnAccess

```
package usagecontrol;
import javax.swing.UIManager;
import accessobject.*;
import components.image;
/**
 * @author KESHNEE
 *
 * TODO To change the template for this generated type comment go to Window -
 * Preferences - Java - Code Style - Code Templates
 */
public aspect IntertypeDeclarationOnAccess {
    declare parents: Access implements Runnable;
    Thread Access.aThread;
    public void Access.endrequest() {
        aThread = null;
        close();
    }
    public void Access.run() {
        aThread = Thread.currentThread();
        UIManager.put("swing.boldMetal", Boolean.FALSE);
        image.createAndShowGUI(getObject());
        // Keep going as long as myThread is the same as the current thread.

        while (image.WindowOpen) {
            try {
                Thread.sleep(500); // Tell the thread to sleep for half
            }
            catch (InterruptedException e) {}
        }

        if (!image.WindowOpen){
            endrequest();
        }
    }
}
```

```

    }
    return false;
}
}

```

Source code for OngoingConditions

```

package usagecontrol;
import javax.swing.*.*;

import accessobject.*;
/**
 * @author PADAYK
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */

public class OngoingConditions extends Access implements Runnable{
    /**
     * Controls actions relating to the conditions of access
     */
    private static long condition = 0;
    private volatile boolean stop = false;
    public OngoingConditions(String SubjectName, String ObjectName, String AccessType ) {
        super(SubjectName,ObjectName,AccessType);
    }

    // This will terminate the run() method.
    public void conditionsWarning(){
        ImageIcon icon = new ImageIcon("c:\\icons\\warn1.gif");
        String message = "<html> <font color=blue> "+ subjectName
            +", is <font color = red> PROHIBITED<font color=blue> "
            +"from accessing client file: " + objectName + " after working hours";
        JOptionPane.showMessageDialog(null, message , "CONDITIONS WARNING",
            JOptionPane.INFORMATION_MESSAGE, icon);
    }

    public long getCondition(){
        condition++;
        return condition;
    }

    public boolean conditionisValid(){
        condition++;
        if (condition%10 == 0)
            return false;
        else
            return true;
    }
    public void run() {
        Thread aThread = Thread.currentThread();
        while(!stop && conditionisValid()){
            try {
                Thread.sleep(1000); // Tell the thread to sleep for a second.
            }

            catch (InterruptedException e) {}
        }
        if (!stop){
            conditionsWarning();
        }
    }
    public void endOngoingConditions(){
        stop = true;
    }
}

```

Source Code for OngoingObligations

```
import java.awt.Color;
import javax.swing.ImageIcon;

import components.ImageFrame;
import accessobject.Access;
import accessobject.AccessInformation;

public class OngoingObligations extends AccessInformation implements Runnable{
    /**
     * Controls actions relating the OngoingObligations of the Access
     */
    private ImageFrame OngoingObligationsRequest;
    public OngoingObligations(String SubjectName, String ObjectName, String AccessType) {
        super(SubjectName,ObjectName,AccessType);
    }

    public void run() {
        Thread aThread = Thread.currentThread();
        String Message = "<html><font color = green>" + subjectName+ " ACCESSING...client file:
        "
        + objectName + " WITH RIGHTS "+ accessType+". <br> ";
        Message.toUpperCase();
        ImageIcon icon = new ImageIcon("c:\\files\\OBS.jpg");
        OngoingObligationsRequest = new ImageFrame(600,300,400,400,"Ongoing
        Obligations",Message,icon);
        OngoingObligationsRequest.setForeground(Color.BLUE);
        OngoingObligationsRequest.setResizable(false);
        // Keep going as long as myThread is the same as the current thread.
        System.out.println("Obligations Window"+objectName);
        while (OngoingObligationsRequest.windowOpen()) {
            try {
                Thread.sleep(500); // Tell the thread to sleep for half a second.
            }
            catch (InterruptedException e) {}
        }
        endOngoingObligations();
    }
    public void endOngoingObligations(){
        if (OngoingObligationsRequest.windowOpen()){
            OngoingObligationsRequest.close();
        }
    }
}

    stop = true;
}
```

Source code for class UsageControlInjector

```
package usagecontrol;

import javax.swing.JOptionPane;
import javax.swing.ImageIcon;
import testutilities.*;
import accessobject.Access;
import components.CheckBox;

public aspect UsageControlInjector {
    private static boolean accessOpen;
    private static boolean precondition = true;
    private Thread obligationsThread;
    private Thread conditionsThread;
    private Thread accessThread;
    private Access accessObject;
    private OngoingConditions OnConditions;
    private OngoingObligations OnObligations;
    private String SubjectName;
    private String ObjectName;
    private String AccessType;

    pointcut Intercept_Request(Access AccessObject):
```

```
execution(* *.request(..) && !within(UsageControlInjector)
&& target(AccessObject) );
void around (Access AccessObject ): Intercept_Request(AccessObject){
    Runtime runtime = Runtime.getRuntime();
    MemoryUsage.printUsage(runtime);
    accessOpen = true;
    accessObject = AccessObject;
    SubjectName = accessObject.getSubjectName();
    ObjectName = accessObject.getObject();
    AccessType = accessObject.getAccessType();
    if (OptimisticRights()){
        if (preObligations(SubjectName, ObjectName, AccessType))
        {
            if (preConditions(SubjectName, ObjectName, AccessType)
            || breakTheGlass(SubjectName, ObjectName, AccessType)){
                accessThread = new Thread(AccessObject);
                accessThread.start();

                Onobligations = new OngoingObligations( SubjectName, ObjectName, AccessType);
                obligationsThread = new Thread(Onobligations);
                obligationsThread.start();

                OnConditions = new OngoingConditions(SubjectName,ObjectName,AccessType);
                conditionsThread = new Thread(OnConditions);
                conditionsThread.start();

                while(accessOpen){
                    try {
                        Thread.sleep(500);
                    } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                }
                postObligations(SubjectName,ObjectName,AccessType);
            }
        }
        MemoryUsage.printUsage(runtime);
    }

public boolean OptimisticRights(){
    //determine whether this information is subject to optimistic access control
    return true;
}

// when there is conditions warning
pointcut OngoingCondition(): call(* *.conditionsWarning(..) ) && target(OngoingConditions);
after(): OngoingCondition(){
    initiateBreakTheGlassFacility();
}

// when the access ends
pointcut OngoingAccess() : call(* *.endrequest(..) ) && target(Access) &&
!within(UsageControlInjector);
after(): OngoingAccess(){
    postAccess();
}

// when the user ends ongoingobligations
pointcut OngoingObligation() : call(* *.endOngoingObligations(..) ) &&
target(OngoingObligations) && !within(UsageControlInjector);
after(): OngoingObligation(){
    postAccess();
}

void initiateBreakTheGlassFacility(){
    if (!breakTheGlass(OnConditions.getSubjectName(), OnConditions.getObject(),
OnConditions.getAccessType())){
        Onobligations.endOngoingObligations();
        postAccess();
    }
}
}
```

```
boolean preObligations(String SubjectName, String ObjectName, String AccessType){

    ImageIcon icon = new ImageIcon("c:\\icons\\hand.gif");
    String message = "<html>" + SubjectName + ", if you click <font color=green> YES " +
        "<font color=black> you agree to <font color=red>NOT<font color=black>" +
        " distribute client file: "
        + ObjectName;
    int answer = JOptionPane.showConfirmDialog(null, message,"PRE-OBLIGATIONS",
        JOptionPane.YES_NO_OPTION,JOptionPane.WARNING_MESSAGE,icon);
    if (answer == JOptionPane.YES_OPTION) {
        return true;
    }
    else if (answer == JOptionPane.NO_OPTION) {
        return false;
    }
    return false;
}

boolean preConditions(String SubjectName, String ObjectName, String AccessType){

    if (preCondition) {
        ImageIcon icon = new ImageIcon("c:\\icons\\warn1.gif");
        String message = "<html> <font color=blue>" + SubjectName + ", "
            +" <font color=red>PROHIBITED<font color = blue> from accessing client file: " +
            ObjectName + " at this time !" ;
        JOptionPane.showMessageDialog(null, message ,"PRE-CONDITIONS WARNING",
            JOptionPane.INFORMATION_MESSAGE,icon);

        preCondition = false;
        return false;
    }
    return true;
}

void postAccess(){
    accessOpen = false;
    accessObject.endrequest();
    Onobligations.endOngoingObligations();
    OnConditions.endOngoingConditions();
    //update logs
}

public void postObligations(String SubjectName, String ObjectName, String AccessType){
    CheckBox.createAndShowGUI();
}

boolean breakTheGlass(String SubjectName, String ObjectName, String AccessType){
    BreakTheGlass breakTheGlass = new BreakTheGlass(SubjectName,ObjectName,AccessType);
    if (breakTheGlass.display()){
        logAccess(SubjectName,ObjectName,AccessType,"Illegal Access", "YES");
        return true;
    }
    return false;
}

void logAccess(String SubjectName, String ObjectName, String AccessType, String Notice,
String RedFlag){
    //WRITE TO LOG FILE
}
}
```

APPENDIX D:

PROTOTYPE EVALUATION

D1. Research Methodology

The Design Science Research method was applied. This methodology involves problem identification, design and development, and an evaluation.

D.2 Problem Statement

To validate the concept elucidation of the OAC(UCON) model via evaluative prototyping and to determine if the OAC(UCON) model is perceived as an effective countermeasure against data misuse.

D.3 Design and Development

The product concept was implemented by the researcher using both object-oriented and aspect-oriented techniques. However, in order to remove researcher bias, the product concept was introduced to 14 Honours students at the University of Pretoria. They were not shown the working version as to not to bias their judgement of the concept. Furthermore, as the model was intended to scale up to a real-world scenario, the product specification was placed in a context where optimistic access control enhanced with usage control was viewed within a traditional role-based access control with trust measures. The participants were not expected to use the aspect-oriented approach, as this concept is not currently taught in the syllabus.

Participants were given the following specification as a term assignment:

Using an appropriate open source database application, you will create a simple database to store information about a typical organisation with employees and clients to be serviced. A client record includes inter alia the name, occupation, employer, address, contact details and account details of the client. The employee record includes inter alia the name, salary and period of employment of employees. You will use SQL statements to query the database.

For collaborative purposes the client information is relegated to the public domain, while the employee data is protected by role-based access controls. You need to enforce mixed initiative access controls when the user attempts to access the database.

If the user attempts to access data in the public domain, then he/she is subject to the following usage control mechanisms:

Pre-obligation: The user must click on a button in a dialog box, thereby indicating that he/she agrees not to distribute this information.

Pre-condition: This information may be accessed during business hours only.

Ongoing obligation: A window with the following warning “This dataset must be used EXCLUSIVELY for work-related purposes” is to remain open while the user accesses the information.

Ongoing condition: This information may be accessed during business hours only (same as pre-condition, as it is time dependent). While the pre-condition may have been valid at the time of access, it may become invalid during the access.

Post-obligation: The user must send an e-mail to the administrator if he accessed these databases outside of business hours.

Break-the-glass (BTG): While the user will not be permitted to access the information unless the obligations have been satisfied, he/she will under special circumstances be allowed to access it by utilising the BTG facility even if the pre-conditions or ongoing conditions are invalid.

Post-update: A user’s rights to information in the public domain can be modified based on prior usage. Your program should log all access in such a way that there is a secure audit

trail. At the onset each user has a trust level of *high*. However, as they demonstrate their untrustworthiness, the level is downgraded to *medium* and finally to *low*. As their trust level drops, they lose their rights to information in the public domain – i.e. information that they are allowed to access is limited. Users with a *medium* trust level can access most information except for account information. Users with a *low* trust level are not allowed to view account information or contact details. They can be limited to view less sensitive details such as the client's name, occupation, etc.

After the user has accessed the database, his/her trust level is updated by using fuzzy logic. For test purposes, each access can be given a random priority [0, 1]. If the BTG facility was deployed by the user, then the trust level [0,1] is updated, dependent on the priority of the task and the user's previous trust level using the fuzzy matrix given below.

| Priority of Task \ Previous Trust Level | High | Medium | Low |
|---|-----------------------------------|---|---|
| High | Trust level remains High | Trust level downgraded to Medium | Trust level downgraded to Medium |
| Medium | Trust level remains Medium | Trust level downgraded to Low | Trust level downgraded to Low |
| Low | Trust level remains Low | Trust level remains Low | Trust level remains Low |

The employee records are protected by role-based access controls. In this system there are three roles, namely *manager*, *administrator* and *user*. The manager can *read*, *delete* and *update* an employee record, whereas an administrator can *read* and *update* an employee record. Users can *read* employee records, but all salary information is concealed.

You need to authenticate users (by means of passwords) and stipulate access control policies for the data in the database. If you prefer Java as your language of implementation,

you could use the Java Sandbox model [9] to authenticate users and stipulate access control policies for the data in the database.

You will need to create a policy file to grant permissions to authenticated users.

You will need to create a login configuration file for authentication.

D.4 Evaluation

This small-scale experiment will test the theory that users' interaction with the prototype will be perceived as an effective countermeasure against data misuse. In order to test the hypothesis, two qualitative data collections will be employed, namely participant observation and open-ended interviewing.

Observation: The idea with participant observation is to determine whether access control measures were implemented successfully.

Qualitative Interview: A qualitative interview is to be conducted to determine the developer's perceptions of the appeal of the prototype in terms of data misuse.

Qualitative Questionnaire: Participants were asked to address the following in terms of the model concept: Weaknesses, Strengths, Improvements, Viability, Applicability, Scalability.

EVALUATION

Indicate whether you agree or disagree with the following statements and give reasons for your answer.

| Statements | Agree/ Disagree |
|--|---------------------------|
| The product specifications as given in the assignment were ambiguous and incomplete. | Agree [] Disagree [] |
| Reason: | |
| The product specifications as given in the assignment could easily be translated into an implementable product. | Agree [] Disagree [] |
| Reason: | |
| In terms of the enforcement of security, other mechanisms such as a written policy document or adequate training would have been more effective than the mechanisms identified in the product concept. | Agree [] Disagree [] |
| Reason: | |
| The flexibility offered under the optimistic access control domain is a security risk. | Agree [] Disagree [] |
| Reason: | |
| Specifying system conditions – such as limiting access according to the time of day – may deter users from abusing their privileges. | Agree [] Disagree [] |
| Reason: | |
| The 'Break-the-Glass' facility is vulnerable to abuse. | Agree [] Disagree [] |
| Reason: | |
| The protection mechanisms – such as fulfilling obligations – will compel users to comply with the established rules of behaviour in order to protect confidential information. | Agree [] Disagree [] |
| Reason: | |
| An individual who interacts with the system will recognise that access is dependent on user responsibility as well as technical access control. | Agree [] Disagree [] |
| Reason: | |
| The risk of losing one's rights to information under the optimistic access control domain may deter one from abusing one's privileges. | Agree [] Disagree [] |
| Reason: | |
| The conditions, obligations and the Break-the-Glass mechanisms may be distracting to a user. | Agree [] Disagree [] |
| Reason: | |



| | |
|--|-----------------------------------|
| Most users will ignore the messages in terms of the conditions and obligations relating to access. | Agree <input type="checkbox"/> |
| | Disagree <input type="checkbox"/> |

Reason:

Any other comments or critique

D.5 Overview of the Experiment-and-evaluation-of-use study and its application

Sessions

Several separate sessions were held at the laboratory of the School of Computer Science. Each session was attended by one participant and lasted approximately 30 minutes. Prior to their individual session, participants were given the concept specification to implement as a term assignment. Each participant then had to demonstrate his/her working product and provide value judgements on the model concept.

Steps involved

The participants were given the following test cases to carry out in order to interact with the prototype, while the researcher observed:

1. An authorised user accessing data under the public domain is allowed to read client data but is subject to pre-conditions, pre-obligations, ongoing obligations, ongoing conditions and post-obligations (simulate instances where the pre-conditions and ongoing conditions are not satisfied).
2. An unauthorised user is not able to access any data.
3. An authorised user is subject to role-based access control policies when accessing the employee data.
4. User's optimistic rights are downgraded.

The participants were then asked several open-ended questions regarding the usability and perceived effectiveness of the product as a security countermeasure [see questions in D.4 above].

Validity

Postgraduate students were used to develop and evaluate the model concept in view of the fact that they have extensive knowledge in information systems and are currently employable. Seeing that some of these students are already employed within the



information systems sector, their profile can serve as a profile of user representatives in systems development.

APPENDIX E: DATA COLLECTION

| Statement1: The product specifications as given in the assignment were ambiguous and incomplete. | | | |
|--|--|-------|----------|
| Participant | Responses | Agree | Disagree |
| A | In fact, I just followed the instructions incorporating minimal creativity from class discussions | | YES |
| B | It was sufficient | | YES |
| C | It was relatively open-ended but not incomplete | | YES |
| D | I was able to implement it early | | YES |
| E | Much was left to (mis) interpretation | YES | |
| F | I think it was good but the concept of priorities should be stated more | | YES |
| G | All the aspects of the system has been explained clearly | | YES |
| H | To a certain extent because it did not give explicit rules of the break the glass | YES | |
| I | NO RESPONSE | | YES |
| J | RBAC and optimistic access control were clearly specified except for how priority is assigned | | YES |
| K | The assignment was straightforward | | YES |
| L | NO RESPONSE | YES | |
| M | All specifications were easy to understand and implement into a system | | YES |
| N | I found the project brief very complete, although I think the system would need more to be commercially viable | | YES |

Statement 2: The product specifications as given in the assignment could easily be translated into an implementable product.

| Participant | RESPONSE | Agree | Disagree |
|-------------|---|-------|----------|
| A | Well, I wouldn't really know, not really an expert on access control in commercial environments | YES | |
| B | It was properly adapted for the size and situation | YES | |
| C | NO RESPONSE | YES | |
| D | NO RESPONSE | YES | |
| E | No Issues | YES | |
| F | Yes I think it can be used in certain types of industry | YES | |
| G | Since it was enough to implement the prototype, given that it is scaled up, the real product could be implemented | YES | |
| H | yes but it can only scale to certain levels | YES | |
| I | NO RESPONSE | YES | |
| J | Specification was detailed enough | YES | |
| K | It is close to that point upon completion | YES | |
| L | NO RESPONSE | YES | |
| M | Specifications were easy to divide into implementable components | YES | |
| N | Yes, if tested properly, and robust for users to define new conditions | YES | |

Statement 3: In terms of the enforcement of security, other mechanisms such as, a written policy document or adequate training would have been more effective than the mechanisms identified in the product concept.

| Participant | Response | Agree | Disagree |
|-------------|--|-------|----------|
| A | Usage based as implemented in the project is just kinda[sic] weak, Maybe if the break the glass facility were removed entirely | YES | |
| B | PB: They are simpler to ignore | | YES |
| C | The other mechanisms added to the current mechanisms can make the overall product more effective but,.. | | YES |
| D | A companies trust of their employees should have some evaluation criteria | | YES |
| E | Depends on the environment. Ideally both should be used | | YES |
| F | Inforcing[sic] it this way I think would be more effective | | YES |
| G | NO RESPONSE | | YES |
| H | Written policy can be present but they[sic] is not constant reminder like in a automated system | | YES |
| I | NO RESPONSE | | YES |
| J | Learning curve maybe too steep for the average people, requiring training | YES | |
| K | If used in addition it would be more secure | YES | |
| L | With the prototype concept it enables administrators to track user actions | | YES |
| M | The main security risk still lies with the user and no amount of training can truly provide security against human stupidity | | YES |
| N | No personal ethics would still be basis for decisions made by personnel. An agreement etc. would be acceptable in court | | YES |

| Statement 4: The flexibility offered under the optimistic access control domain is a security risk. | | | |
|--|--|-------|----------|
| Participant | Response | Agree | Disagree |
| A | It really all depends on what you are guarding and if the users are being monitored | YES | |
| B | Never give a user more slack than the minimum | YES | |
| C | it might, depending on the level of confidentiality of the data | YES | |
| D | The users might see and distribute sensitive data | YES | |
| E | Should be controlled through other means | YES | |
| F | In some cases I think letting people access data with BTG can cause major harm | YES | |
| G | The information can be abused with optimistic access control | YES | |
| H | Because it is subject to evaluation by a human | | YES |
| I | It can be because they is no monitor to ensure that risk free activities are done | YES | |
| J | Too much data is allowed to be viewed | YES | |
| K | NO RESPONSE | | YES |
| L | But it depends on the nature of the organisations and its data | YES | |
| M | If used intelligently a lot of information can be accessed without dire consequences | YES | |
| N | No certain environment require that flexibility such the medical industry | | YES |

| Statement 5: Specifying system conditions, such as limiting access according to the time-of-day, may deter users from abusing their privileges. | | | |
|--|---|-------|----------|
| Participant | Response | Agree | Disagree |
| A | The threat of losing trust would deter me, if I really wanted to get in I would rather just try avoid the access control system | YES | |
| B | It may, the key being "may" | YES | |
| C | NO RESPONSE | | YES |
| D | Error dialogues might frighten some uses and deter them from continuing | YES | |
| E | Depends on the consequences of violating them | YES | |
| F | If people want to abuse their privileges they can do it during work hours | | YES |
| G | Since ignoring the restrictions would lead to degradation of trust | YES | |
| H | Yes but what about a situation when a company works overtime | YES | |
| I | NO RESPONSE | YES | |
| J | As long as the user is aware of these issues it will deter them | YES | |
| K | Users should be less likely to abuse their privileges during working hours | YES | |
| L | It will give a user a feeling that they are doing something wrong | YES | |
| M | A person would mostly try to be unseen when doing misconduct and that the easiest after hours | YES | |
| N | No, not if defined correctly and the BTG functionality provides alternatives | | YES |

| Statement 6: The 'break the glass' facility is vulnerable to abuse. | | | |
|--|---|--------------|-----------------|
| Participant | Responses | Agree | Disagree |
| A | Most definitely in my implementation it relied mostly on the threat of an admin stepping in after the event | YES | |
| B | NO RESPONSE | YES | |
| C | The trust level drops too quickly | | YES |
| D | The trust level drops | | YES |
| E | Necessary nonetheless | YES | |
| F | Yes, I think some user will abuse the BTG feature | YES | |
| G | Not if the task at hand is of low priority | | YES |
| H | Because you can use it to get back at employee when di[sic] | YES | |
| I | Everything is logged and the user will be monitored | | YES |
| J | The priority of requests need to be better refined | YES | |
| K | To the extent that everything is vulnerable to abuse | YES | |
| L | NO RESPONSE | YES | |
| M | As stated above, if used correctly a lot of information can be accessed without dire consequences | YES | |
| N | No, not if the manager/auditor does not abuse the system by allowing any situation or accepting any reason | YES | |

| Statement 7: The protection mechanisms, such as fulfilling obligations, will compel users to comply with the established rules of behaviour in order to protect confidential information. | | | |
|--|---|--------------|-----------------|
| Participant | Responses | Agree | Disagree |
| A | Nobody reads EULAs | | YES |
| B | Well it does | YES | |
| C | It might, do a user test | YES | |
| D | This would have to be tested as the users think and act differently | | YES |
| E | Depends on the implementation of those obligations | YES | |
| F | If not doing degrades their access right it will compel the users to comply with it | YES | |
| G | NO RESPONSE | YES | |
| H | NO RESPONSE | YES | |
| I | NO RESPONSE | YES | |
| J | Users are intimidated by warning usually | YES | |
| K | It adds a sense of responsibility | YES | |
| L | But only if the user is trustworthy. | YES | |
| M | Because ignorance is not an excuse anymore | YES | |
| N | Yes | YES | |

| Statement 8: An individual who interacts with the system will recognize that access is dependent on user responsibility as well as technical access control. | | | |
|---|---|-------|----------|
| Participant | Responses | Agree | Disagree |
| A | The warning should be a clear indication | YES | |
| B | NO RESPONSE | YES | |
| C | This depends on the level of knowledge the user have of the trust-based system | YES | |
| D | The trust level indication | YES | |
| E | Don't assume users are responsible | | YES |
| F | As a user uses it for some time. I think he will get accustomed to responsible usage of the sys | YES | |
| G | NO RESPONSE | YES | |
| H | Yes, because the constant prompts | YES | |
| I | NO RESPONSE | YES | |
| J | Technical side may be obvious e.g. audit logs | | YES |
| K | See above | YES | |
| L | They might notice that their amount of actions they can performs degrades | YES | |
| M | This will only be the case for the majority if it is explained clearly at the beginning | | YES |
| N | Yes | | YES |

| Statement 9: The risk of losing one's rights to information under the optimistic access control domain may deter one from abusing one's privileges. | | | |
|--|--|-------|----------|
| Participant | Responses | Agree | Disagree |
| A | Although it depends what their ultimate goal is. I.e. if they are planning on quitting the next day, they should even care | YES | |
| B | Depends on the information content and nature. Also user registration | * | |
| C | NO RESPONSE | YES | |
| D | If the information is vital to the user | YES | |
| E | Only if the information is wanted but not needed | | YES |
| F | If not fulfilling obligation reduces their access rights they will be careful with the use of data | YES | |
| G | Since the user can't see sensitive info | YES | |
| H | Yes fear is always a deterrent | YES | |
| I | NO RESPONSE | YES | |
| J | Lets the user know when can go wrong and violate privileges | YES | |
| K | NO RESPONSE | YES | |
| L | If the user's goal is to steal data, it won't prevent them from doing so | | YES |
| M | If your actions are logged that caused it , you have a higher possibility of being caught | YES | |
| N | Yes | YES | |

* - NO RESPONSE

Statement 10: The conditions, obligations and the break-the-glass mechanisms may be distracting to a user.

| Participant | Responses | Agree | Disagree |
|-------------|---|-------|----------|
| A | It depends on the user, I say disagree because I suspect most users will simply ignore the mechanisms and they will lose their meaning anyway | | YES |
| B | If they want the info. they won't mind | | YES |
| C | It is quite distracting at times, as all error messages/info messages are. | YES | |
| D | At first, last they might ignore it User testing again is vital here | YES | |
| E | Depends on the implementation of UI | YES | |
| F | To some extent if they are many | YES | |
| G | NO RESPONSE | | YES |
| H | Constant popups are distracting to the user | YES | |
| I | It might be a bit strange at first but the user should be able to get use to it | | YES |
| J | This is needed to deter users | | YES |
| K | NO RESPONSE | | YES |
| L | It will only effect them when they log in and go past business hours | | YES |
| M | Windows vista used a similar approach with permission popups and must users disabled this security feature due to annoyance | YES | |
| N | PN: If the conditions are important enough, it should not | | YES |

Statement 11: Most users will ignore the messages in terms of the conditions and obligations relating to the access.

| Participant | Responses | Agree | Disagree |
|-------------|--|-------|----------|
| A | Once they learn which sequence of buttons to press to get to the required result, why would they bother reading? Or caring for that matter | YES | |
| B | True, do you ever read the agreement when you start | YES | |
| C | Depends on what they know about the consequences of ignoring them | | YES |
| D | If the user is forced to give response to the message | | YES |
| E | Will after repeated exposure but then users will know its contents | | YES |
| F | NO RESPONSE | YES | |
| G | NO RESPONSE | | YES |
| H | Yes until they are warned that they will loose[sic] access | YES | |
| I | They might at first but once they notice that is limits their access after it will be taken seriously | | YES |
| J | Most sensible users will feel threaten by the messages | | YES |
| K | Initially they will pay heed, but later it becomes routine | YES | |
| L | Most users will just want to get the data to do their work | YES | |
| M | This will only happen if the consequences are not clearly specified | | YES |
| N | Yes but the responsibility still lies with the them and holds them accountable | YES | |

APPENDIX F:

ASPECTJ SEMANTICS

The following list of pointcut designators describe only those designators that were used in the context of thesis:

| | |
|--|---|
| call (signature) execution (signature) | Matches call/execution joinpoints at which the method or constructor matches the signature |
| target (ClassName) | All the join points where the object on which the method called is an instance of ClassName |
| within (className) | matches join points of any kind at which the currently executing code is contained with ClassName |
| declare parents: ClassName implements InterfaceName | declares the ClassName type to implement the InterfaceName Interface |

Wildcards

Type names that contain the two wildcards "*" and ".." are also type patterns. The * wildcard matches zero or more characters except for ".", so it can be used when types have a certain naming convention.

Operators

Pointcuts compose through the operations **or** ("|"), **and** ("&") and **not** ("!")

Cited from:

1. Kiczales, G., et al. *An Overview of AspectJ*. In *ECOOP '01: Proceedings of the 15th European Conference on Object-Oriented Programming*. 2001. Budapest, Hungary: Springer-Verlag. p. 327-353
2. <http://www.eclipse.org/aspectj/doc/released/progguide/semantics-pointcuts.html>
(Last accessed 1 October 2009)

APPENDIX G: RUNNING DEMO PROJECT

Go to:

<http://cs-cert.unisa.ac.za/internet/keshnee/content.html>

Instructions for Running Demo:

- (1) Create a directory called **test0** on your **C** drive
- (2) Unzip [test0.zip](#) to into **test0** directory
- (3) Search for `aoptest.exe` in directory **test0** and double click to run
- (4) Read [Manual.pdf](#) for more details on the operation of the software

This software was built using:

Aspect J version 1.4.0

Eclipse version 3.2

Java SDK 1.4.2_05